修士学位論文

論文題目　Android OSを搭載した組み込みデバイスとスマートフォンによるモータ制御

氏名（ふりがな）　齋藤　翼（さいとう　つばさ）

専攻　工学研究科　機械工学専攻

指導教授　金丸　隆志　准教授

修了年月（西暦）　2012年　3月

工学院大学大学院

目次

概要

現在，Android OS が注目を集めており，スマートフォン用 OS としてだけでなく，組み込み端末用 OS としても利用され始めている．本研究では，Android OS の利用法の 1 つとしてロボットアームの制御を提案する. スマートフォンと Android OS を搭載させた組み込みデバイスをターゲットとしており, スマートフォンのセンサ値を Bluetooth で送信し，それを受信した組み込みデバイスがロボットアームの制御を行う．センサ値はそのままでは制御に適さないため，ニューラルネットワークで加工してから送信する．これによりスマートフォンから組み込みデバイスによるロボットアームの制御が実現できた.

1章　序論

1-1 研究背景

今や誰もが手にしている携帯電話は，1996 年頃から徐々に普及し始め，2011 年 6 月の時点で普及率は 94.7%，総契約台数は 1 億 2124 万台となっている(図 1-1)[1].

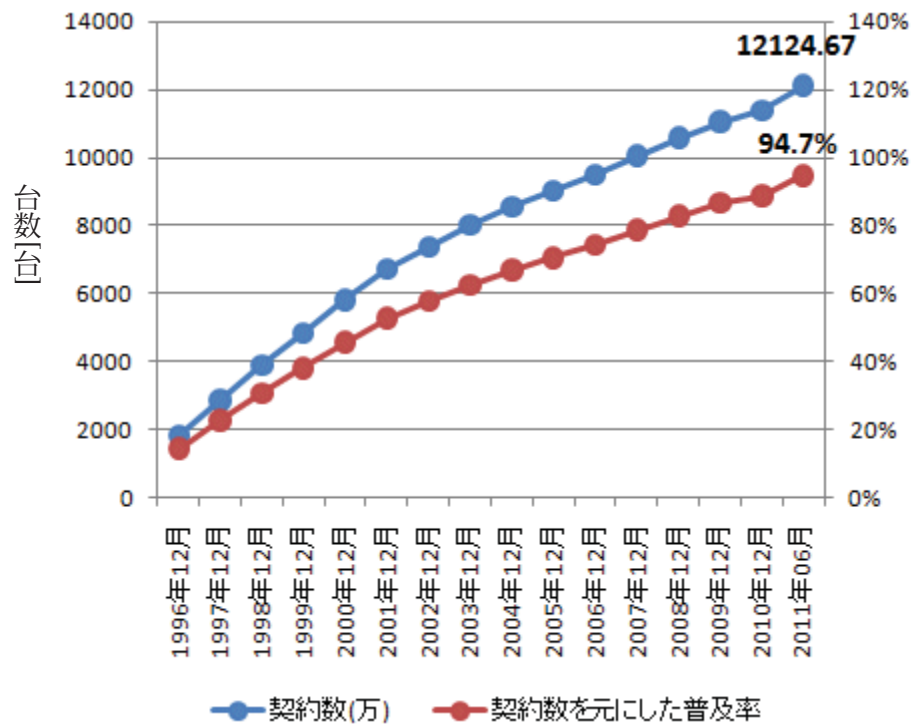



図 1-1 日本での携帯電話・自動車電話契約数と契約数を元にした普及率[1]

単純計算すると，現在日本で生活しているほとんどの人が携帯電話を所持していることになる．近年スマートフォンの登場により，この携帯電話業界に大きな転機が訪れた．スマートフォンとはインターネットとの親和性が高く，パソコンの機能をベースとして作られた多機能携帯電話である．その変化を顕著にみられるデータとして携帯電話の出荷台数の割合の推移が図 1-2 に示されている．

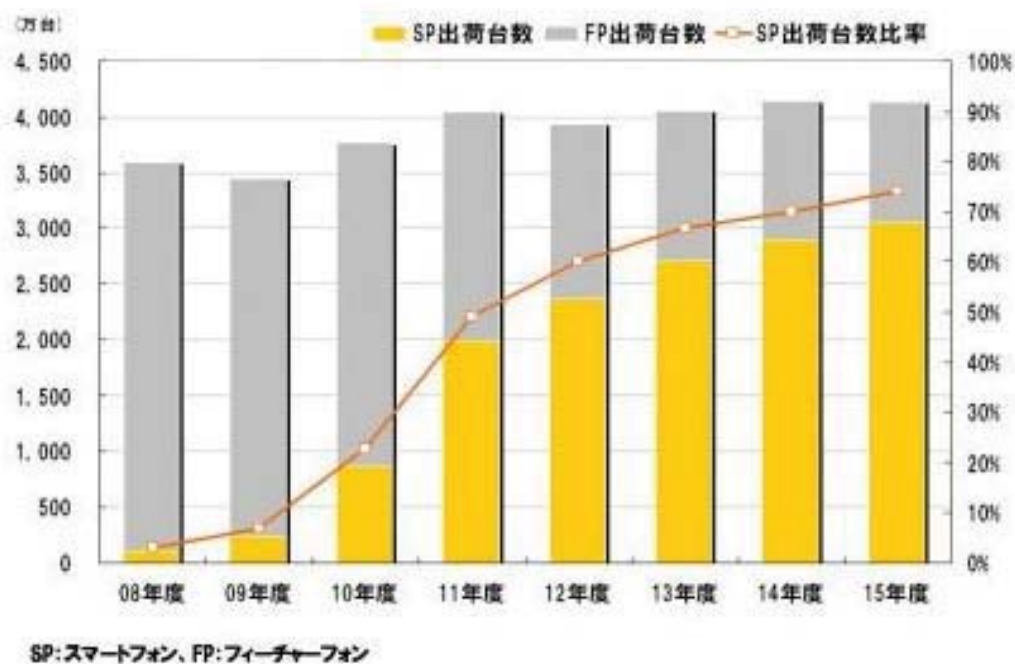



図 1-2 スマートフォン出荷台数・比率の推移・予測(2011 年 3 月現在)[2]

2009 年度から 2010 年度にかけてスマートフォンの出荷台数は約 3 倍になり，さらに 2010 年度と 2011 年度にかけて約 2 倍に増えている.その大きな理由として挙げられるのが 2008 年に発売されたアップル社の iPhone が人気を集めたことと，2010 年に Android OS 搭載スマートフォン「Xperia」が登場したことである.

Android OS はアメリカの Google が中心となって組織された Open Handset Alliance によって開発された携帯電話やタブレット端末用の OS である．Android OS が広まった大きな理由の一つは，オープンソースであり誰でも無償で利用することができたことであると考えられる．他のスマートフォン向け OS に比べオープン性が高く，ライセンスのコストも必要ない．また，携帯電話の高機能化に従って開発コストが急上昇してきたことから，それが抑えられるということで, 携帯電話会社が揃って Android OS を搭載したスマートフォンの開発に取り組み始めた．これにより，2011 年 11 月現在でスマートフォン用の OS としては日本でトップシェアとなった(図 1-3)[3].

図 1-3 Android OS と iOS のシェア数推移[3]

Android はアプリケーションの開発も容易に行える為, 現在では個人でアプリケーションの開発を行うユーザーも増えている．それにより，2011 年 8 月にアプリケーションの総数は約 28 万個(図 1-4)[4]，アプリケーションダウンロード総数は 100 億を突破した．

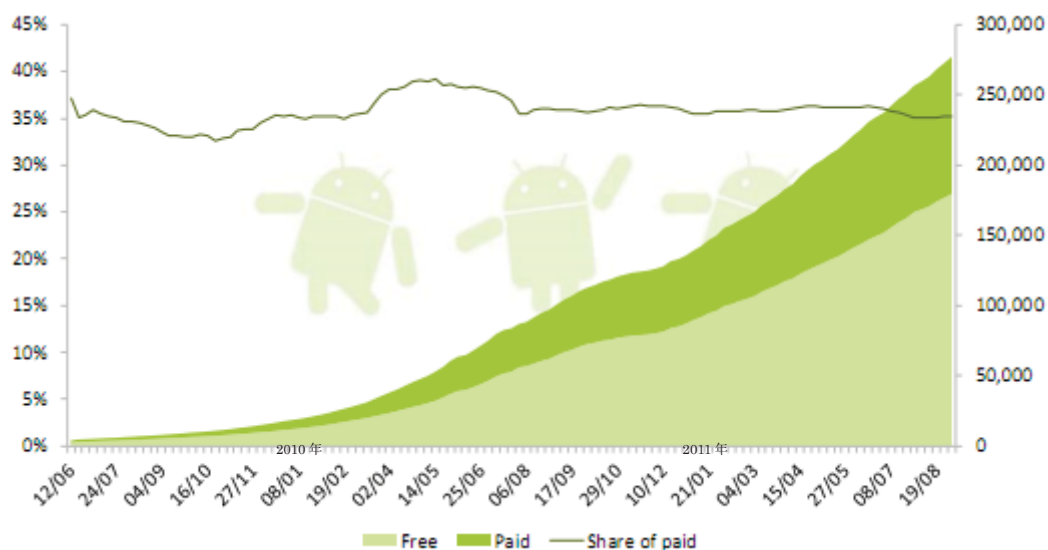



図 1-4 Android マーケットにおけるアプリケーション数[4]

また, Android OS は汎用性が高く, 現在では携帯電話用 OS としてだけではなく, 家電,

自動車のカーナビゲーションシステムなどにも利用され始めている．図 1-5 は Android OS を搭載したテレビである．

図 1-5 Android OS 搭載テレビ

このように，汎用性の高い Android OS をスマートフォンとは別の組み込みデバイス用 OS として利用することで，Android OS の利用の場を増やせるのではないかと筆者は考えた．

1-2 Android について

Android はスマートフォンやタブレット PC などの携帯電話端末をターゲットとしたプラットフォームである．2007 年 11 月に Google，米クアルコム社，独通信キャリアの T-Mobile International 社などが中心となり設立した「Open Handset Alliance」が Android を発表した．2009 年に日本では初の Android OS 搭載(Android OS 1.5)のスマートフォン「HT-03A」が NTT ドコモから発売された(図 1-6)．

図 1-6 HT-03A

Android 搭載スマートフォンが日本で発売されてから 2，3 年の間に OS のバージョンは 1.5，1.6，2.1，2.2，2.3，4.0 と更新され，新しい機能が次々に追加された．バージョンによる変更点の一部を表 1-1 に挙げる．

表 1-1 Android のバージョンアップによる機能の追加

OS バージョン	追加機能
1.6	Google マップの機能向上
	音声検索対応
	デザイン向上による操作性の向上
2.1	複数の Google アカウントの登録が可能
	ライブ壁紙の対応
	ピンチ操作(2 つの指を広げたり縮めたりする操作)の対応
2.2	Adobe Flash10.1 の対応
	テザリング対応
	アプリケーションの外部メモリに保存可能
	JIT コンパイラの導入
2.3	端末操作性の向上・高速化
	ゲーム関連機能の強化
	NFC(おサイフケータイ機能)への対応
	バッテリー消費の改善
4.0	ブラウザの高速化
	顔認証機能
	データ通信料の管理
	スクリーンショットの標準搭載

1-3 研究目的

本研究では Android OS の新たな利用法として，スマートフォンと Android 搭載組み込みシステムを利用したロボットアームの遠隔制御の実現を目指す．

我々が注目したのは，Android に搭載されたセンサデバイスである．温度センサ，近接センサなど多くのセンサが利用できる中から 3 方向加速度センサ，傾きセンサを利用する．これらのセンサ値を加工した値を Bluetooth 送信するスマートフォンアプリケーションと，モータ制御する Android 搭載組み込みデバイスによりロボットアームの遠隔制御を実現する． Android OS を搭載させる組み込みデバイスのターゲットとして，TEXAS INSTRUMENTS 製の BeagleBoard および PandaBoard を選択した． Android のバージョンは Android2.2.2(Froyo)，Android4.0.1(Ice Cream Sandwich)とする．スマートフォンなどの小型端末と組み込み端末との協調動作が行えることにより，Android OS の利用価値を高め，更なる発展につながるのではないかと考えている．

1-4 論文の構成

本論文の構成を以下に示す．

1 章では本研究に至る背景，目的について述べる．Android OS の歴史や新しい利用法などを示す．

2 章では本研究に用いた開発環境の説明をする．Android OS の構造，組み込みデバイスとして利用する BeagleBoard，PandaBoard の詳細スペックなどを示す．更に，組み込みデバイスに載せる Android のビルド法について解説する．

3 章では本研究で利用したモータの詳細と Java によるモータの制御法を解説する．

4 章では本研究で用いるバックプロパゲーション(ニューラルネットワーク)の理論について解説する．

5 章では作成した Android アプリケーションによるロボットアーム制御について解説する．取得したセンサ値を加工して Bluetooth 送信するスマートフォンアプリケーションと受信した値を基にモータ制御を行う組み込みデバイス用アプリケーションについて解説する．

6 章では本論文の結論を述べる．

2章 組み込みデバイスの環境設定

本章では，モータの制御を行う組み込みデバイスの環境設定法について解説する．

2-1 開発環境

本研究で利用した機材などのリストは表 2-1 の通りである．

表 2-1 開発環境一覧

環境	製品名
組み込みデバイス	BeagleBoard Rev.C4, PandaBoard Rev.A2
OS	Android OS 2.2.2（Froyo），4.0.1（IceCreamSandwitch）
サーボモータ	双葉電子工業製モータ RS-304MD
タッチパネルモニタ	Haniwha 製 7 インチタッチパネル HM-TI7T
シリアル変換器	双葉電子工業製 シリアル変換器 RSC-U485
Linux 搭載 PC	Ubuntu 10.04

ここでは，組み込みデバイス，ターゲットとした Android OS，モータについて説明していく．

2-2 組み込みシステム

2-2-1 BeagleBoard

BeagleBoardは2008年にTEXAS INSTRUMENTSから発売されたマイコンボードである．基盤の外寸サイズは 75[mm] × 75[mm]，プロセッサに OMAP3(ARMv7 Cortex-A8)720MHz，メモリ256MBを搭載し，小型ながら高い性能を持っている．外部インターフェースも豊富で，HDMI，SDカードスロット，USB2.0ポートなどが搭載されている(図 2-1).

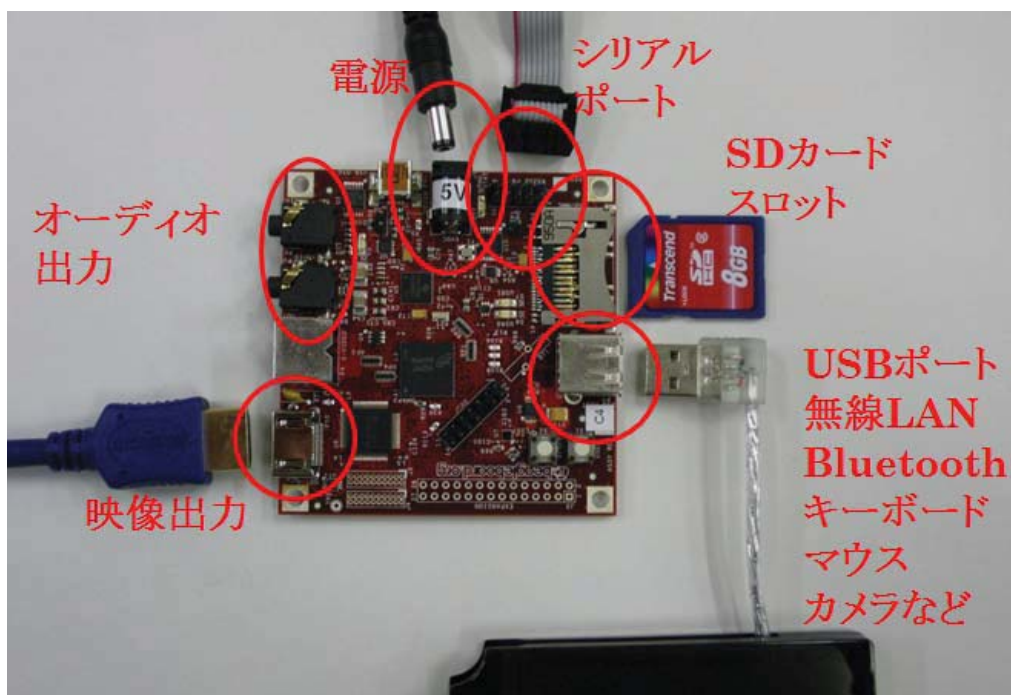



図 2-1 BeagleBoardの外観

2-2-2 PandaBoard

PandaBoardは2011年12月にTEXAS INSTRUMENTSから発売されたBeagleBoardの後継機である．本体サイズ114.3[mm]×101.6[mm]とBeagleBoardよりひとまわり大きくなったが，内蔵メモリが1GBとなり，プロセッサもOMAP4430(1.0GHz)が搭載されている(図2-2).

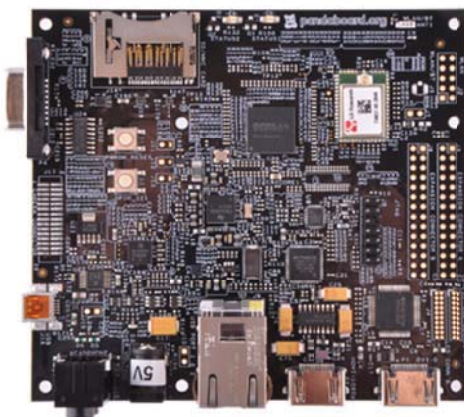



図 2-2 PandaBoard

2-3 Android の構造

Android は kernel とライブラリ，ランタイムのほとんどが C，C++で書かれており，アプリケーション，アプリケーションのフレームワークは Java で書かれている．OS の構造は，図 2-3 のようになっている．Android OS を組み込み用の OS として利用するためにはソースをダウンロードし，その組み込みデバイス用に kernel やソースの改変をする必要がある．Android OS の設定法を以下で簡単に紹介する．

kernel の変更…Bluetooth とシリアルデバイスの有効化

Android ソースの変更…kernel の変更を Android に対応させる．

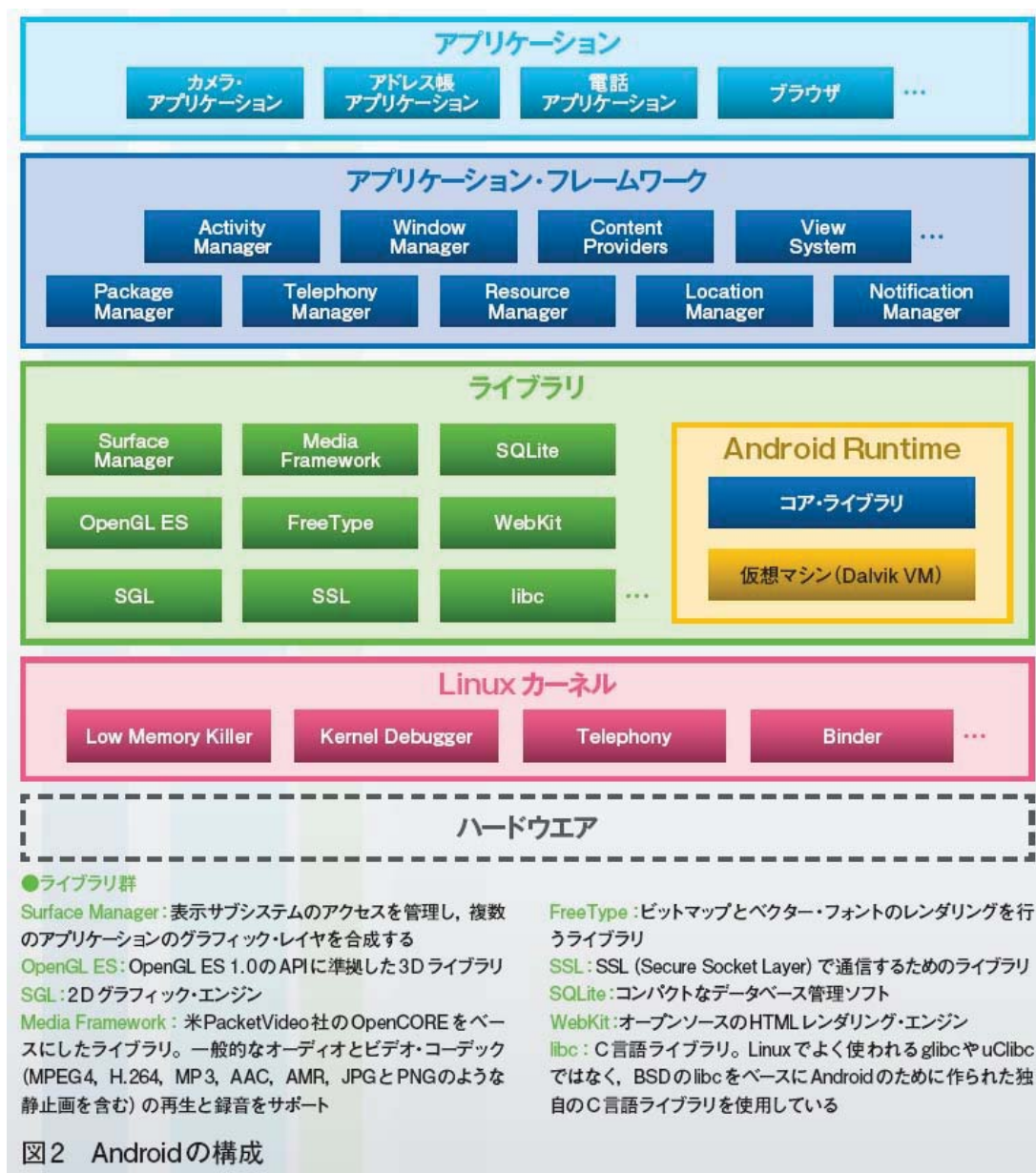



図 2-3 Android の構成[5]

2-4 Android のビルド

本章では Android を BeagleBoard にポーティングする手順を記す．Android OS を BeagleBoard にポーティングする際，TEXAS INSTRUMENTS による公式のもの[6]をそのまま利用することもできるが，多くのデバイスが有効になっていない．そこで，シリアル通信や無線 LAN，タッチパネルを利用するためにソースの変更を行った．その際， sola 氏のホームページ [7]を参考にした．また，ソースの改変はすべて Linux 上で行った．変更内容を以下で簡単にまとめる．

2-4-1 ビルド環境の設定

Linux 端末を用いて，Android のビルドを行う環境の設定を行う．環境設定の内容は以下の通りである．

・各種コマンドのインストール
・Java5 のインストール
・Java6 のインストール
・git/repo の設定
・パスの追加

環境設定の詳細は以下の通りである．

はじめに，以下のコマンドにて Android 開発に必要なソフトをインストールする．

```
$ sudo apt-get install git-core gnupg flex bison gperf libsdl-dev
libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev
zlib1g-dev
$ sudo apt-get install valgrind
$ sudo apt-get install libreadline5-dev
$ sudo apt-get install uboot mkImage
```

次に Java5 をインストールする. 下記のコマンドでファイルを開く. 尚, この方法は Ubuntu に限定される.

```
$ sudo gedit /etc/apt/sources.list
```

下記 2 行を追加する．

```
deb http://us.archive.ubuntu.com/ubuntu/hardy multiverse
deb http://us.archive.ubuntu.com/ubuntu/hardy-updates multiverse
```

追加した後，以下の 2 コマンドにて Java5 のインストールが完了する．

```
$ sudo apt-get update
$ sudo apt-get install sun-java5-jdk
```

インストールが完了した後，先ほど追記した 2 行を削除する．

別バージョンの Java をインストールしている場合，以下のコマンドで変更する．

```
$ sudo update-alternatives --config java
$ sudo update-alternatives --config javac
```

次に下記コマンドで Java6 のインストールを行う．

```
$ sudo add-apt-repository “deb http://archive.canonical.com/ lucid
partner”
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk
```

Java5 は Android 2.2 のビルドの際に利用し，Java6 は Android2.3 以降のビルドの際に利用する．

2-4-2 git/repo の設定

git/repo の設定は以下の通りである．

```
$ mkdir ~/bin
$ curl http://android.git.kernel.org/repo >~/bin/repo
$ chmod a+x ~/bin/repo
```

ここで，「.bashrc」に以下の 1 行を追加する．

```
export PATH=$PATH:~/bin
```

git コマンド 2 つを行う．

```
$ git config --global user.email “メールアドレス”
$ git config --global user.name “ユーザー名”
```

次にパスを追加する．

```
$ vi ~/.bashrc
#.bashrc
# User specific aliases and function
alias rm=’rm -i’
alias cp=’cp -i’
alias mv=’mv -i’
# Source global definitions
if[ -f /etc/bashrc ]; then
./etc/bashrc
Fi

PATH=”$PATH”:~/bin
```

環境設定は以上である．これを基に 2-4-3 から Android のソースの取得とビルドについて記す．

2-4-3 Android のソース取得，ビルド

2-4-1，2-4-2 で設定したビルド環境を用いてビルドを行っていく．

2-4-3-1 SD カードの設定

BeagleBoard は SD カードから Android をブートする．SD カードは以下のように 3 つのパーティションに分ける．

DISK1：ファイルシステム fat32，容量 64MB 程度，kernel 格納用．
DISK2：ファイルシステム fat32，容量 DISK1，DISK3 の残り．Android から SD カードとして見える．
DISK3：ファイルシステム ext3，容量 1～2GB 程度，Android のシステム領域

2-4-3-2 ソースの準備

Android のソースをダウンロードする．初めにソースをダウンロードするディレクトリを下記のコマンドで作成する．今回は android-2.2.2_r1 というディレクトリを作成した．

```
$ mkdir android-2.2.2_r1
```

作成した android-2.2.2_r1 ディレクトリに移動し，このディレクトリを ANDROID という環境変数に設定する．

```
$ cd android-2.2.2_r1
$ export ANDROID=`pwd`
```

次に repo スクリプトファイルを以下の引数で実行し，リポジトリの初期化を行う．

```
$ repo init -u https://android.googlesource.com/platform/manifest
```

ここで Your Name，Your Email の入力を要求されるので，適切な情報を入力する．

```
repo initialized in /home/ユーザー名/android-2.2.2_r1
```

と表示されれば完了である．次に ALSA 関連のファイルを追加する．

```
$ gedit .repo/local_manifest.xml
```

local_manifest.xml を開き．以下の内容を記述する．

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <project path="external/alsa-lib"  name="platform/external
    /alsa-lib" revision="froyo"/>
  <project path="external/alsa-utils" name="platform/external
    /alsa-utils" revision="froyo"/>
  <project path="hardware/alsa_sound" name="platform/hardware
    /alsa_sound" revision="froyo"/>
</manifest>
```

local_manifest.xml を保存した後，ソースコードを取得する．repo スクリプトファイルを以下の引数で実行し，ソースコードを取得する．

```
$ repo sync
```

Android のソースのダウンロードについては以上である．

次に，kernel のダウンロードと展開を行う．以下の 3 コマンドで$ANDROID に kernel_beagle.0xlab.sgx.tar.bz2 を展開する．

```
$ cd $ANDROID
$ wget http://sola-dolphin-1.net/data/android/BeagleBoard
  /kernel_beagle.0xlab.sgx.tar.bz2
$ tar jxvf kernel_beagle.0xlab.sgx.tar.bz2
```

BeagleBoard 用の変更ファイル，追加ファイルを下記コマンドにて適用する．

```
$ cd $ANDROID
$ wget http://android-development-environment.googlecode.com
  /files/vendor_sola-omap3-froyo.tar.gz
$ mkdir vendor
$ tar zxvf vendor_sola-omap3-froyo.tar.gz -C $ANDROID/vendor/
$ $ANDROID/vendor/sola/omap3/patch/omap3-patch.sh
```

次に，筆者が所属する研究室のホームページ[8]に公開されている vender_tk-beagle-froyo2.2.x-20110725.tar.gz を以下のコマンドにてダウンロードし，パッチの適用をする．

```
$ cd $ANDROID
$ wget http://brain.cc.kogakuin.ac.jp/research/files
  /vendor_tk-beagle-froyo2.2.x-20110725.tar.gz
$ tar zxvf vendor_tk-beagle-froyo2.2.x-20110725.tar.gz -C $ANDROID/vendor/
$ $ANDROID/vendor/tk/patch/tk_patch.sh
```

2-4-3-3 kernel の変更

本研究で利用するデバイスを有効化する方法についてまとめる．2-4-3-2 で行った vender_tk-beagle-froyo2.2.x-20110725.tar.gz の適用により有効になる設定やデバイスは以下の通りである．

・日本語フォントのデフォルトインストール

・画面の常時点灯設定

・機内モードの削除

・各種デバイスの有効化(Bluetooth，USB シリアル，USB カメラ，無線 LAN，タッチスクリーン)

ここでは研究で使用した Bluetooth とシリアルの有効化について記載し，他のデバイスの有効化については付録で記載する．kernel の設定ファイルは

$ANDROID/kernel-beagleboard/arch

/arm/configs/sola_omap3_beagle_android_defconfig である．

この内部の変更を行うことでこれらのデバイスが有効になる．

2-4-3-4 Bluetooth の有効化

Bluetooth の有効化に関する変更は以下の通りである.

```
CONFIG_BT=y
CONFIG_BT_L2CAP=y
CONFIG_BT_SCO=y
CONFIG_BT_RFCOMM=y
CONFIG_BT_RFCOMM_TTY=y
CONFIG_BT_BNEP=y
CONFIG_BT_HIDP=y
CONFIG_BT_HCIBTUSB=y
CONFIG_USB_ARCH_HAS_OHCI=y
CONFIG_INPUT_UINPUT=y
```

$Android/system/bluetooth/bluedroid/Android.mk の include$(CLEAR_VARS)の次の行に以下の 3 行が挿入されている.

```
ifeq ($(HAVE_NO_RFKILL_SWITCH),true)
       LOCAL_CFLAGS += -DNO_RFKILL_SWITCH
endif
```

$ANDROID/system/bluetooth/bluez/src/adapter.c の

add_rfcomm_service_record 関数内で unit8_t channel;が以下のように変更されている.

```
//       uint8_t channel;
       uint16_t channel;
```

$ANDROID/vendor/sola/beagleboard/BoardConfig.mk の末尾に以下の 2 行が追加されている.

```
BOARD_HAVE_BLUETOOTH := true
HAVE_NO_RFKILL_SWITCH := true
```

$ANDROID/vendor/sola/beagleboard/init.rc にてコメントアウトされた下記の部分の行頭の#をすべて削除して有効にする．ただし，以下には本当のコメントの 2 行が含まれているので，その#は削除しないよう注意する必要がある(init.rc does not yet…の部分).

```
#service dbus /system/bin/dbus-daemon --system --nofork
#    socket dbus stream 660 bluetooth bluetooth
#    user bluetooth
#    group bluetooth net_bt_admin

#service bluetoothd /system/bin/bluetoothd -n
#    socket bluetooth stream 660 bluetooth bluetooth
#    socket dbus_bluetooth stream 660 bluetooth bluetooth
   # init.rc does not yet support applying capabilities, so run as
root and
   # let bluetoothd drop uid to bluetooth with the right linux
capabilities
#    group bluetooth net_bt_admin misc
#    disabled

(中略)

#service opush /system/bin/sdptool add --channel=12 OPUSH
#    user bluetooth
#    group bluetooth net_bt_admin
#    disabled
#    oneshot

#service pbap /system/bin/sdptool add --channel=19 PBAP
#    user bluetooth
#    group bluetooth net_bt_admin
#    disabled
#    oneshot
```

system/bluetooth/bluedroid/bluetooth.c が vender_tk に含まれているファイルで差し替えられている．Bluetooth に関する変更は以上である．

2-4-3-5 USB-シリアルデバイスの有効化

USB 経由のシリアルデバイスを有効化に関する変更は以下の通りである．尚，本研究で有効にしたシリアルデバイスは Elecom 製 UC-SGT と Futaba 製モータ制御用デバイス RSC-U485 である．

```
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_FTDI_SIO=y
CONFIG_USB_SERIAL_PL2303=y
```

ここで，FTDI_SIO は RSC-U485 を利用するためのドライバ，PL2303 は UC-SGT を利用するためのドライバである．UC-SGT についてはこの変更のみで利用できるようになるが，RSC-U485 はさらに下記 2 ファイルを編集する必要がある．

```
$ANDROID/kernel-beagleboard/drivers/usb/serial/ftdi_sio.h
$ANDROID/kernel-beagleboard/drivers/usb/serial/ftdi_sio.c
```

ftdi_sio.h には RATOC のデバイスの後ろに 2 行が挿入されている．

```
#define RATOC_VENDOR_ID         0x0584
#define RATOC_PRODUCT_ID_USB60F 0xb020

#define FUTABA_VID                     0x1115  /* inserted by TK */
#define FUTABA_RSCU485_PID      0x0008  /* inserted by TK */
```

ftdi_sio.c には下記の位置に 1 行挿入されている.

```
{ USB_DEVICE(FTDI_VID, FTDI_DOMINTELL_DGQG_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_DOMINTELL_DUSB_PID) },
{ USB_DEVICE(FUTABA_VID, FUTABA_RSCU485_PID) },  /* inserted by TK
*/
{ },                                  /* Optional parameter entry */
```

以上が kernel の変更内容である.

2-4-3-6 kernel のビルド

kernel の変更がすべて終わった後，以下のコマンドを行うことにより kernel が作成される.

```
$ cd $ANDROID/kernel-beagleboard
$ make ARCH=arm CROSS_COMPILE=../prebuilt/linux-x86/toolchain
/arm-eabi-4.4.0/bin/arm-eabi-
sola_omap3_beagle_android_defconfig
$ make ARCH=arm CROSS_COMPILE=../prebuilt/linux-x86/toolchain
/arm-eabi-4.4.0/bin/arm-eabi- uImage modules
```

kernel のビルドが終了した後，下記のディレクトリにある uImage をパーティション分けした SD カードの DISK1 にコピーする.

```
$ANDROID/kernel-beagleboard/arch/arm/boot/uImage
```

2-4-3-7 Android のビルド

下記の 4 コマンドにて Android のビルドを行う．Android2.2 をビルドする場合，Java のバージョンが 5 でないとビルドすることができない.

```
$ cd $ANDROID
$ source build/envsetup.sh
$ lunch beagleboard-eng
$ make -j8
```

ビルドが完了した後，下記コマンドで Android のイメージを作成する.

```
$ cd $ANDROID
$ $ANDROID/vendor/sola/omap3/image/beagleboard-image.sh
```

次に下記コマンドで TI's Android SGX SDK を組み込む．この際，インストール先を，この際インストール先を$ANDROID/TI_Android_SGX_SDK とする.

```
$ cd $ANDROID
$ git clone git://gitorious.org/rowboat/ti_android_sgx_sdk.git
TI_Android_SGX_SDK
$ cd TI_Android_SGX_SDK
$ ./OMAP35x_Android_Graphics_SDK_setuplinux_3_01_00_03.bin
```

次に$ANDROID/TI_Android_SGX_SDK/Rules.make を編集する.

```
$gedit $ANDROID/TI_Android_SGX_SDK/Rules.make
```

上記コマンドで指定ファイルを開き，下記の 5 行を追加する.

```
HOME=$(ANDROID)
GRAPHICS_INSTALL_DIR=$(ANDROID)/TI_Android_SGX_SDK
ANDROID_ROOT=$(ANDROID)/vendor/sola/omap3/image/beagleboard/android
CSTOOL_DIR=$(ANDROID)/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/
KERNEL_INSTALL_DIR=$(ANDROID)/kernel-beagleboard
```

編集，保存後，下記の 3 コマンドを行うことで $ANDROID/vendor/sola/omap3/image/beagleboard/android に必要なファイルがコピーされる.

```
$ cd $ANDROID/TI_Android_SGX_SDK
$ make
$ make install OMAPES=3.x
```

$ANDROID/vendor/sola/omap3/image/beagleboard/android 中身を SD カードの DISK3 にコピーし，下記コマンドにて SD カード内を誰でも書き換えを行えるようにする.

```
sudo chmod -R 777 /media/DISK3
```

以上の操作により，Android システムの改変，ビルドが完了した.

2-5 BeagleBoard の構成を変更

2-4-3-1 にて SD カードを順に kernel/SD(保存領域)/Android システムのようにパーティション分けを行ったが BeagleBoard のブート設定では最初に DISK2 を読みこむ設定となっている．そこで DISK3 を最初に読みこむ設定にする必要がある．

ターミナルエミュレータに PuTTY を利用し，BeagleBoard と Windows PC をシリアルで接続する．

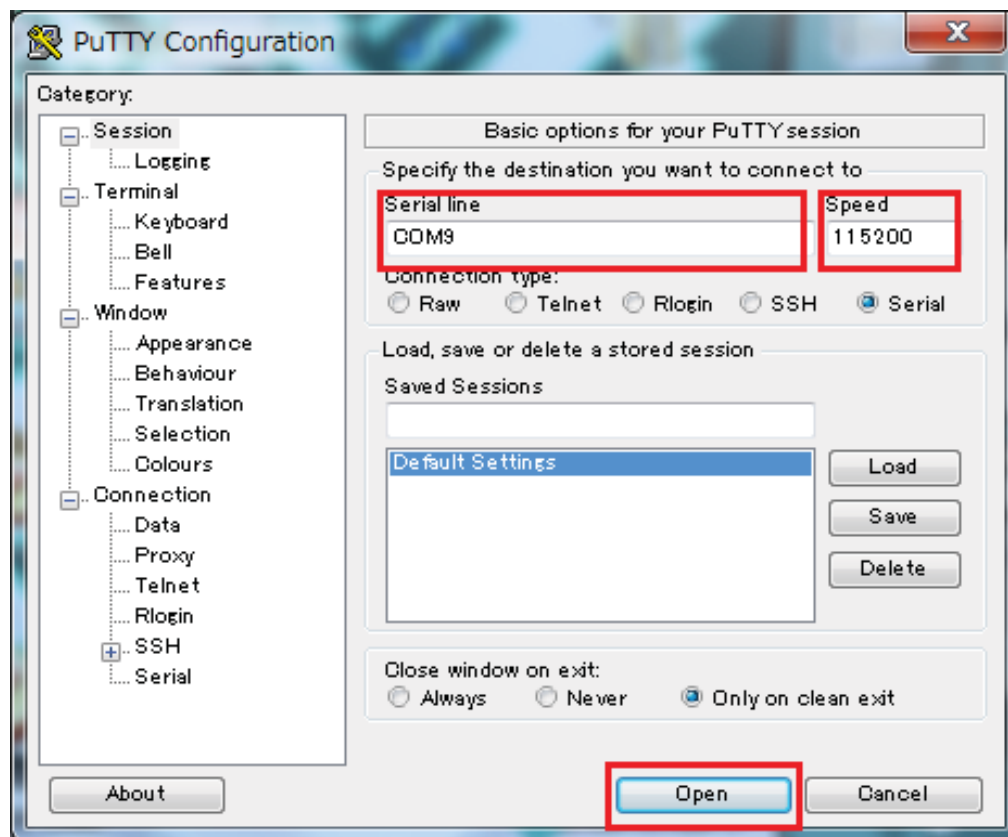



図 2-4 PuTTY の接続設定

PuTTY を起動すると図 2-4 のような設定ウィンドウが表示されるので，起動 Sereal line（ポート）と Speed（接続速度 115200bps）を指定し，「Open」する．ここではシリアルポートは「COM9」が存在するとする．

Open 後，BeagleBoard の電源を入れると図 2-5 のように「Hit any key to stop autoboot」と表示されるので，何かキーを叩く．

図 2-5 PuTTY 操作 1

コマンドで printenv を実行すると環境変数が表示される．ここで編集するのは mmcroot という環境変数である(図 2-6)．

```
COM9 - PuTTY
Hit any key to stop autoboot:  0
OMAP3 beagleboard.org # printenv
bootcmd=if mmc init; then if run loadbootscript; then run bootscript; else if ru
n loaduimage; then run mmcboot; else run nandboot; fi; fi; else run nandboot; fi
bootdelay=10
baudrate=115200
loadaddr=0x82000000
console=ttyS2,115200n8
vram=12M
dvimode=1024x768MR-16@60
defaultdisplay=dvi
mmcroot=/dev/mmcblk0p2 rw
mmcrootfstype=ext3 rootwait
nandroot=/dev/mtdblock4 rw
nandrootfstype=jffs2
nandargs=setenv bootargs console=${console} vram=${vram} omapfb.mode=dvi:${dvimo
de} omapfb.debug=y omapdss.def_disp=${defaultdisplay} root=${nandroot} rootfstyp
e=${nandrootfstype}
```

図 2-6 PuTTY 操作 2

```
setenv mmcroot "/dev/mmcblk0p3 rw init=/init"
```

を入力することで図 2-6 の赤枠部が mmcroot=/dev/mmcblk0p3　rw init=/init に変更される．最後に saveenv のコマンドを入力することで変更が保存される．

3章　モータの制御

3-1 モータについて

3-1-1 モータの仕様

本研究では双葉電子工業製モータ RS-304MD を利用する(図 3-1).

図 3-1 双葉電子工業製モータ RS-304MD

RS-304MD の仕様は以下のとおりである.

表 3-1 RS-304MD 仕様

トルク	5.0[kgf・cm]
スピード	0.16[sec/60°]
重量	21[g]
電源電圧	4.8〜7.4[V]
可動範囲	-150〜150[度]

3-1-2 モータの制御方法

RS-304MD はコマンド方式と PWM 方式どちらの信号でも制御が可能である．本研究では USB-シリアル変換器である RS-U485 を利用しコマンド方式を採用した.

3-1-3 メモリーマップ

RS-304MD は動作のためのデータを保存するメモリ領域を持っており，電源を切ると消えてしまう「RAM 領域」と電源を切っても値を保存できる「ROM 領域』がある．それぞれの領域のメモリーマップは付録に記載する.

3-1-4 サーボ ID

RS-304MD は個々に ID 番号を設定できる．この ID はコマンド方式で動作中にサーボの個体を識別するために付けられた固有の番号である．初期値は 1 になっており，1 つの通信系で複数のサーボを接続する場合は，ID を個々に設定する必要がある．ID の変更については 3-2 にて記載する.

3-1-5 パケット

メモリーマップに対して送信するデータは以下の通りである.

Header	ID	Flag	Address	Length	Count	Data	Sum

・Header

パケットの先頭を表し，FA　AF に設定する.

・ID

サーボ ID，1~127(01H～7FH)までの値が仕様できる．255 に設定すると全 ID のサーボへ共通指令になる.

・Flag

サーボからのリターンデータの取得やデータ書き込み時の設定を行う.

・Address

メモリーマップ上のアドレスを指定する．このアドレスから「Length」に指定した長さ分のデータをメモリーマップに書き込む.

・Count

サーボの数を表す.

・Data

メモリーマップに書き込むデータ.

・Sum

送信したデータの確認用チェックサムで，パケットの「ID」から「Data」の末尾までを 1 バイトずつ XOR した値を設定する.

(例)下記の送信データのチェックサムは以下のようになる.

Header	ID	Flag	Address	Length	Count	Data	Sum
FA AF	01	00	1E	02	01	00 00	1C

3-2 サーボ ID の変更

本研究では複数のモータを同時に利用する．初期のモータのサーボ ID はすべて 1 となっている為，変更する必要があった．そこで ID の変更には双葉電子工業の公式ページの C 言語版サンプルを一部書き換えたものを用いた．ID を 1 から 5 に変更するために送信するパケットは次のようになっている．

Header	ID	Flag	Address	Length	Count	Data	Sum
FA AF	01	00	04	01	01	05	00

(fromID)　　(toID)

以下は上記パケットを利用して ID を変更する関数 IDCh の一部である．

```
int IDCh( HANDLE hComm, unsigned char fromID, unsigned char toID ){
//～～～～～～～～～～～～～～～～中略～～～～～～～～～～～～～～～～
        // ハンドルチェック
        if( !hComm ){
                return -1;
        }
        memset( sendbuf, 0x00, sizeof( sendbuf )); // バッファクリア
        //ID 変更
        sendbuf[0]  = (unsigned char)0xFA;       // ヘッダー1
        sendbuf[1]  = (unsigned char)0xAF;       // ヘッダー2
        sendbuf[2]  = (unsigned char)fromID;     // 古い ID
        sendbuf[3]  = (unsigned char)0x00;       // フラグ
        sendbuf[4]  = (unsigned char)0x04;       // アドレス
        sendbuf[5]  = (unsigned char)0x01;       // 長さ
        sendbuf[6]  = (unsigned char)0x01;       // 個数
        sendbuf[7]  = (unsigned char)toID;       // 新しい ID

        // チェックサムの計算
        sum = sendbuf[2];
        for( i = 3; i < 8; i++ ){
                sum = (unsigned char)(sum ^ sendbuf[i]);
        }
        sendbuf[8] = sum;                     // チェックサム
        PurgeComm( hComm, PURGE_RXCLEAR );   // 通信バッファクリア
        ret = WriteFile( hComm, &sendbuf, 9, &len, NULL );   // 送信
        return ret;
}
```

fromID(現在の ID)と toID(変更したい ID)を変更することで ID の変更ができる．この作業によりモータは変更後の ID で動作するが，電源 OFF になってしまうと戻ってしまう．設定を変更した場合はフラッシュ ROM に書き込む必要がある．フラッシュ ROM に書き込むためのパケットは次のようになっている．

Header	ID	Flag	Address	Lenhgth	Count	Sum
FA AF	05	40	FF	00	00	BA

上記パケットをモータへ送信するためのソースは以下の通りである.

```
int FLASHW( HANDLE hComm, unsigned char ID ){
//～～～～～～～～～～～～～～～～～中略～～～～～～～～～～～～～～～～～～
        // ハンドルチェック
        if( !hComm ){
                return -1;
        }
        memset( sendbuf, 0x00, sizeof( sendbuf ));   // バッファクリ
ア
        // パケット作成
        sendbuf[0]  = (unsigned char)0xFA;       // ヘッダー1
        sendbuf[1]  = (unsigned char)0xAF;       // ヘッダー2
        sendbuf[2]  = (unsigned char)ID;         // サーボ ID
        sendbuf[3]  = (unsigned char)0x40;       // フラグ
        sendbuf[4]  = (unsigned char)0xFF;       //アドレス
        sendbuf[5]  = (unsigned char)0x00;       // 長さ
        sendbuf[6]  = (unsigned char)0x00;       // 個数
        // チェックサムの計算
        sum = sendbuf[2];
        for( i = 3; i < 7; i++ ){
                sum = (unsigned char)(sum ^ sendbuf[i]);
        }
        sendbuf[7] = sum;                    // チェックサム
        // 通信バッファクリア
        PurgeComm( hComm, PURGE_RXCLEAR );
        ret = WriteFile( hComm, &sendbuf, 8, &len, NULL );  // 送信
        return ret;
}
```

変更，保存する数値は以下のコードで変更できる.

```
IDCh(hComm,1,5); // ID 1 を 5 に
Sleep(100);
FLASHW(hComm, 5); // ID 5 を保存
Sleep(1000);
```

3-3 モータの操作

実際にモータ ID が変更されているかなどをチェックするため，シークバーを用いてモータを操作させるアプリケーションを作成した．初めに C#で作成し(図 3-2)，次に Java で作成し(図 3-3)，最後に BeagleBoard に載せた Android アプリケーション(図 3-4)で動作確認を行った．

図 3-2 モータ制御アプリ(C#)

C#で作成したアプリケーションのモータのトルク ON はシリアルポートの接続時に ON にしている．以下はモータのトルクを ON/OFF するためのパケットを送信する RSTorqueOnOff 関数である．

```
private void RSTorqueOnOff(short sMode) {//トルク ON/OFF
            byte sum;
            int i;
            sendbuf[0] = (byte)0xFA;
            sendbuf[1] = (byte)0xAF;
            sendbuf[2] = (byte)Servo_id;
            sendbuf[3] = (byte)0x00;
            sendbuf[4] = (byte)0x24;
            sendbuf[5] = (byte)0x01;
            sendbuf[6] = (byte)0x01;
            sendbuf[7] = (byte)(sMode&0x00FF);

            sum = sendbuf[2];
            for (i = 3; i < 8; i++)
            {
                sum = (byte)(sum ^ sendbuf[i]);
            }
            sendbuf[8] = sum;                         //チェックサム
            serialPort1.Write(sendbuf, 0, 9);
}
```

ここで sMode が 1 の場合モータのトルクが ON になり，0 の場合モータのトルクが OFF になる.

以下は，ポート接続ボタン押下時のイベントハンドラ button2_Click のソースである. ポートを Open した後に RSTorqueOnOff 関数が呼ばれていることがわかる.

```
private void button2_Click(object sender, EventArgs e) {//ポート接続ボタン
            if (serialPort1.IsOpen){
                serialPort1.Close();
            }
            serialPort1.PortName = "COM" + PortText.Text;
            try{
                serialPort1.Open();
                PortState.Text = "接続完了";
                short sMode = 1;
                RSTorqueOnOff(sMode);
            }
            catch (Exception ex){
                MessageBox.Show(serialPort1.PortName
                                + "への接続に失敗しました。"
                                + ex.Message, "エラー",
                                 MessageBoxButtons.OK,
                                 MessageBoxIcon.Error);
                PortState.Text = "未接続";
            }
        }
```

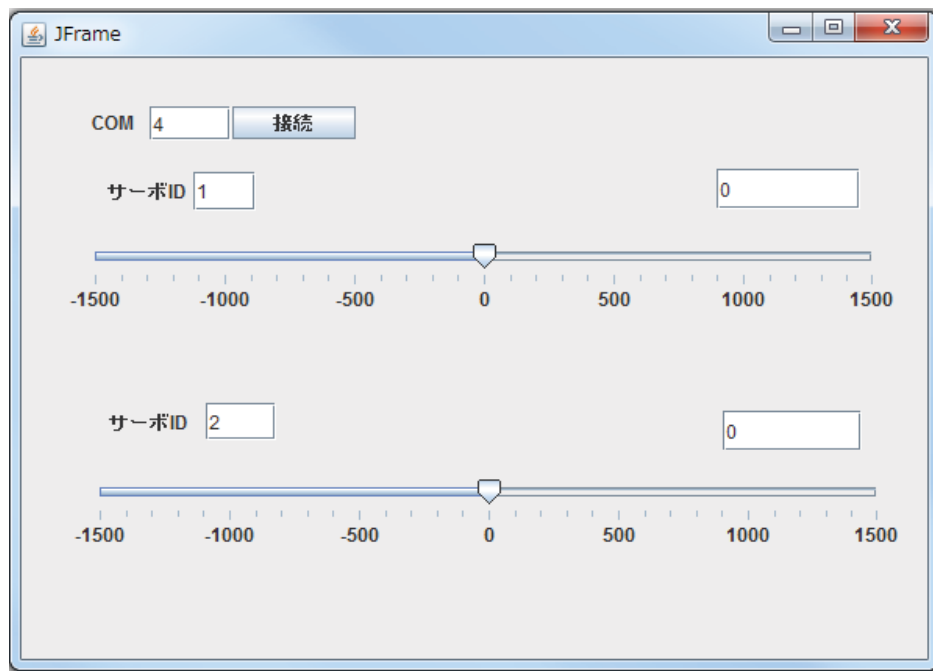



図 3-3 モータ制御アプリ(Java)

Java で作成したモータ制御アプリケーション(図 3-3)も C#でのアプリケーションと同様にシリアルポート接続と同時にモータのトルクを ON にする．Java でシリアル通信を行うにあたり，JNI を利用する RXTX ライブラリを利用した(付録参照).

以下は Java で作成されたポート接続，トルク ON を行う openSerialPort 関数である．

```
boolean openSerialPort() {
        try {
                portId = CommPortIdentifier.getPortIdentifier
                        ("COM" + jTextField1.getText());
                port = (SerialPort)portId.open("VETest", 2000);
                jButton.setText("接続済");
                    setSerialPort();
                    short sMode = 1;
                    RSTorqueOnOff(sMode);
                } catch (NoSuchPortException e) {
                    //e.printStackTrace();
                        System.err.println("NoSuchPort");
                    portId = null;
                    port = null;
                    jButton.setText("接続");
                    return false;
                } catch (PortInUseException e) {
                    //e.printStackTrace();
                        System.err.println("PortInUse");
                    portId = null;
                    port = null;
                    jButton.setText("接続");
                    return false;
                }
        return true;
}
```

図 3-4 モータ制御アプリ(Android)

最後に，Android によるモータ制御アプリケーション(図 3-4)を作成したが，Android アプリケーションはそのままではシリアル通信を行うことができない. そこで JNI(Java Native Interface)を利用する必要がある．Java で作成したアプリケーションでは JNI を利用する外部ライブラリ RXTX を利用したが(付録)，Android アプリケーションでは JNI による自作の C モジュールを作成した．JNI の導入を次節に記す.

3-3-1 JNI の導入

本研究では BeagleBoard でシリアル通信を利用してモータ制御を行う．しかし Java や Android のアプリケーションのみではハードウェアに直接アクセスすることができない為，シリアル通信のために JNI(Java Native Interface)を利用する．JNI は Java のプログラムと C，C++で書かれたコードを連携するためのインターフェースである．JNI の特徴として「ハードウェアにアクセスすることができる」こと，「高速に動作するライブラリをかけること」ことなどが挙げられる．今回は前者を目的として利用することで Java から kernel へのアクセスが可能になる(図 3-5).

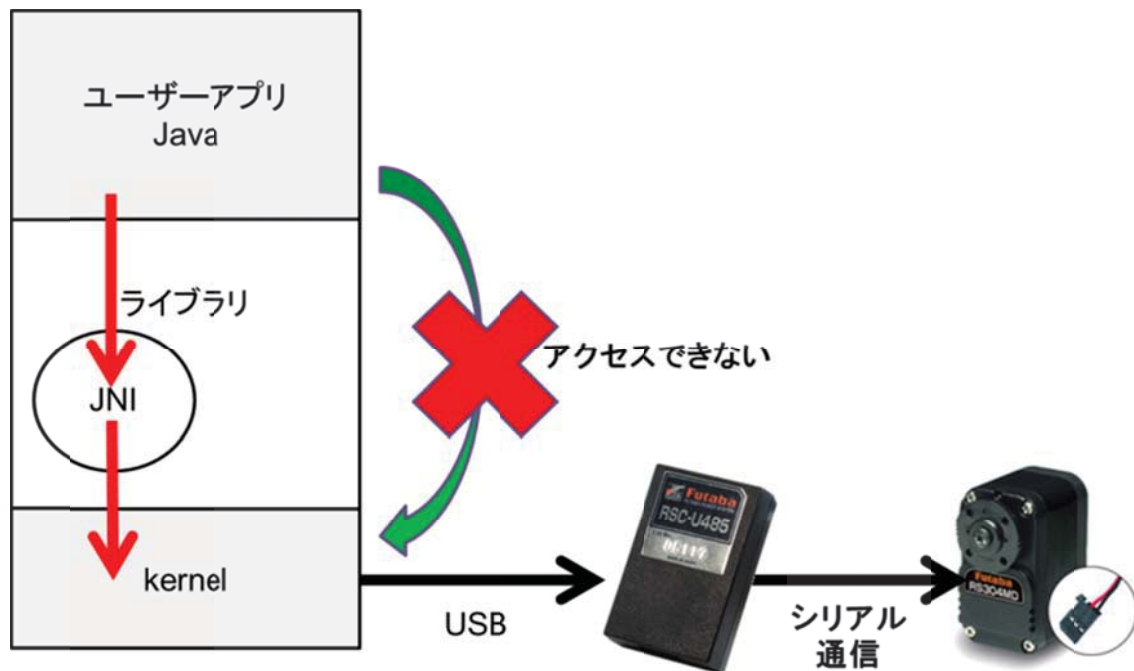



図 3-5 Java(Android)からシリアルでモータの制御

3-3-2 JNI の利用準備

JNI の導入手順は以下の通りである.

・Cygwin のダウンロード, インストール
・Android NDK のダウンロード
・Cygwin でユーザー用スクリプトの作成
・eclipse の設定
・シリアル通信用のモジュールを書き込む
・Cygwin で ndk-build

「本節ではシリアル通信用のモジュールを書き込む」部分について記載する. それ以外の詳細手順は付録に記載する.

3-3-2-1 プロジェクト内に作成した「Android.mk」の編集

eclipse 上で, jni フォルダの中にファイル名「SerialJNI.c」を作成する. ここにシリアル通信のためのモジュールを書き込む. Java へエクスポートする関数は「Java_プロジェクト名_アクティビティ名_関数名」とする. 本研究では
「Java_com_tk_android_BluetoothReceiverKHR1_BluetoothReceiver_openSerial」
となっている. 以下はシリアル通信のモジュールを記載した「SerialJNI.c」の一部である.

```
void
Java_com_tk_android_BluetoothReceiverKHR1_BluetoothReceiver_openSeria
l( JNIEnv* env, jobject thiz, jstring port_s, jint baudrate){
// ポート文字列取得 (/dev/ttyUSB0, /dev/ttyS0, etc)
	const char *port_c = (*env)->GetStringUTFChars(env, port_s, NULL);
	// ボーレート用定数設定 (from /usr/include/bits/termios.h)
	switch(baudrate){
		case 0:
			baud_const = 0000000;
			break;
//～～～～～～～～～～～～～～～～～～～中略～～～～～～～～～～～～～～～～～～～～～
		case 4000000:
			baud_const = 0010017;
		default:
			baud_const = 0000015; // baudrate=9600
	}
	if(fd!=0){
		ioctl(fd, TCSETS, &oldtio);	/* ポートの設定を元に戻す */
		close(fd);			/* デバイスのクローズ */
		fd=0;
	}
	fd = open(port_c, O_RDWR);   /* デバイスをオープンする */
	ioctl(fd, TCGETS, &oldtio);  /* 現在のシリアルポートの設定を待避 */
	newtio = oldtio;		/* ポートの設定をコピー */
	// ボーレート変更
	newtio.c_cflag &=  ~CBAUD;
	newtio.c_cflag |=  baud_const;
	// ビット数変更
	newtio.c_cflag &=  ~CSIZE;
	newtio.c_cflag |=  CS8;
	newtio.c_cflag &=  ~(PARENB | CSTOPB);    // No Parity, Stop bit 1
	// raw mode
	newtio.c_iflag &=~(IGNBRK | BRKINT | PARMRK | ISTRIP
				| INLCR | IGNCR | ICRNL | IXON);
	newtio.c_oflag &= ~OPOST;
	newtio.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG | IEXTEN);
	ioctl(fd, TCSETS, &newtio);	/* ポートの設定を有効にする */
	(*env)->ReleaseStringUTFChars(env, port_s, port_c);
}
```

下記のコードは SerialJNI.c のシリアル通信をクローズさせるための Java_com_tk_android_BluetoothReceiverKHR1_BluetoothReceiver_closeSerial 関数である.

```
void
Java_com_tk_android_BluetoothReceiverKHR1_BluetoothReceiver_closeSeri
al( JNIEnv* env, jobject thiz )
{
	if(fd==-1){
		return;
	}
	ioctl(fd, TCSETS, &oldtio);	/* ポートの設定を元に戻す */
	close(fd);			/* デバイスのクローズ */
	fd=-1;
}
```

下記のコードは，SerialJNI.c のシリアルのデータを書き込むための Java_com_tk_android_BluetoothReceiverKHR1_BluetoothReceiver_writeData 関数である.

```
void
Java_com_tk_android_BluetoothReceiverKHR1_BluetoothReceiver_writeData
( JNIEnv* env,
                                                    jobject thiz , jbyteArray buf,
jint n)
{
        if(fd==-1){
                return;
        }
        jboolean b;
        jbyte *jbuf=(*env)->GetByteArrayElements(env,buf,&b);
        write(fd,jbuf,n);
        (*env)->ReleaseByteArrayElements(env, buf, jbuf, 0);
}
```

Java ソースファイルに以下の 3 行を追加する. これが C モジュールを利用する宣言となる.

```
public native void openSerial(String port, int baudrate);
public native void closeSerial();
public native void writeData(byte[] buf, int n);
```

4章 バックプロパゲーション

スマートフォンで取得したセンサの値を加工するにあたり，本研究ではバックプロパゲーション（誤差逆伝播法)を利用した．バックプロパゲーションは，ニューラルネットワークを訓練するために使われる学習アルゴリズムである[9].

4-1 階層型ネットワーク[10]

図 4-1 のようにニューロンを層状に結合した階層型のネットワークを考える. このネットワークの入力層にデータを入力すると，各ユニットで何らかの計算処理を受けた信号が次の層へと伝わっていき，最終層からある値が出力される．このとき，ユニットとユニットの間には可変な結合強度があり，これらの値が変わることによってネットワークの入出力関係が変わってくる(結合強度を適当に変えることによってさまざまな入出力関係を実現できる)．そこに手本となる入出力データ(教師データ)を繰り返しネットワークに見せると，ネットワークは，教師データが提示されるごとに結合強度を適切に少しずつ調整していき，できるだけその入出力関係をまねようとする．これがネットワークの学習である．

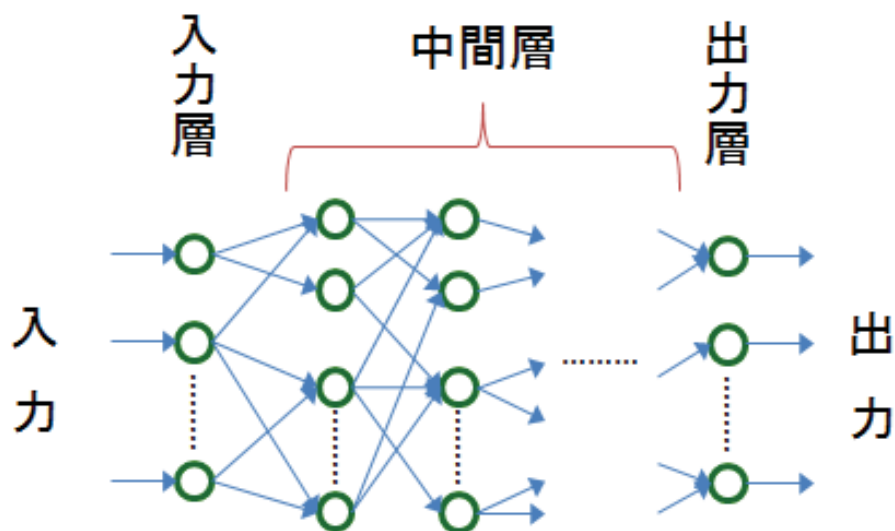



図 4-1 階層型ネットワーク

このようなネットワークの原型といえるものにパーセプトロンがある．与えられた入出力関係がパーセプトロンで実現可能なものであれば，その学習は必ず収束することが知られている．しかし，図 4-2 のような入力層と出力層の 2 層のみからなる単純パーセプトロンでは与えられた問題によってはその入出力関係を実現できない場合がある．このような場合ネットワークを多層にする図 4-1 のように，入力層と出力層の間に中間層を入れることで，ネットワークの識別能力を上げることができる．このような多層パーセプトロンでは，学習は出力ユニットに対してのみ行われ，入力層とし中間層の間の結合はランダムに決める

が，ここの結合も学習によって効率的にできるとよい．このような発想から提案されたものが「バックプロパゲーション」という学習アルゴリズムである．

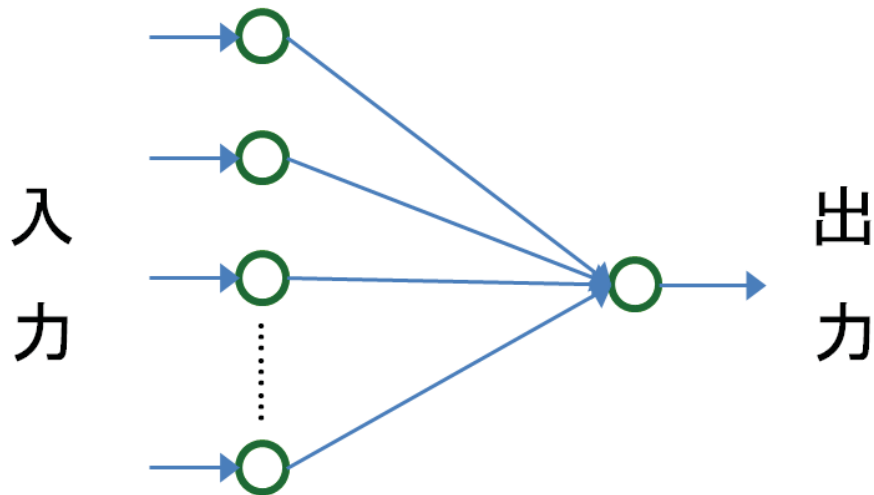



図 4-2 パーセプトロン

ネットワークの構造と各ユニットの働きについて簡単に整理する．ネットワークの層はいくつでもよいが，2 層では単純パーセプトロンと同じになってしまうため，3 層以上と考える．層状ネットワークなので，ある層内にあるユニットはそのひとつ前の層内のすべての(あるいは一部の)ユニットから入力を受け，それに応じた出力を次の層の各ユニットに伝搬する．式で書くと第k層のi番目のユニットは，

$$v_i^k = \sum_j w_{ij}^{k-1} u_j^{k-1} + t_i^k$$

の入力を受け，これに応じて

$$u_i^k = f(v_i^k)$$

の出力を出す．ただしw_{ij}^{k-1}は第$k-1$層第jユニットと第k層第iユニットの間の結合強度であり，t_i^kは第k層第iユニットの閾値である．パーセプトロンでは出力関数$f(v)$が閾値関数(入力vが正のとき 1 を出し，0 以下のとき 0 を出す)を用いる．バックプロパゲーションを用いるためには，この$f(v)$は入力vで微分できるものでなければならない．そこで微分でき，更に閾値関数によく似た関数として，次のような関数が用いられる．

$$f(v) = \frac{1}{1 + \exp(-v)}$$

この関数は，シグモイド関数と呼ばれ図 4-3 のようになる．

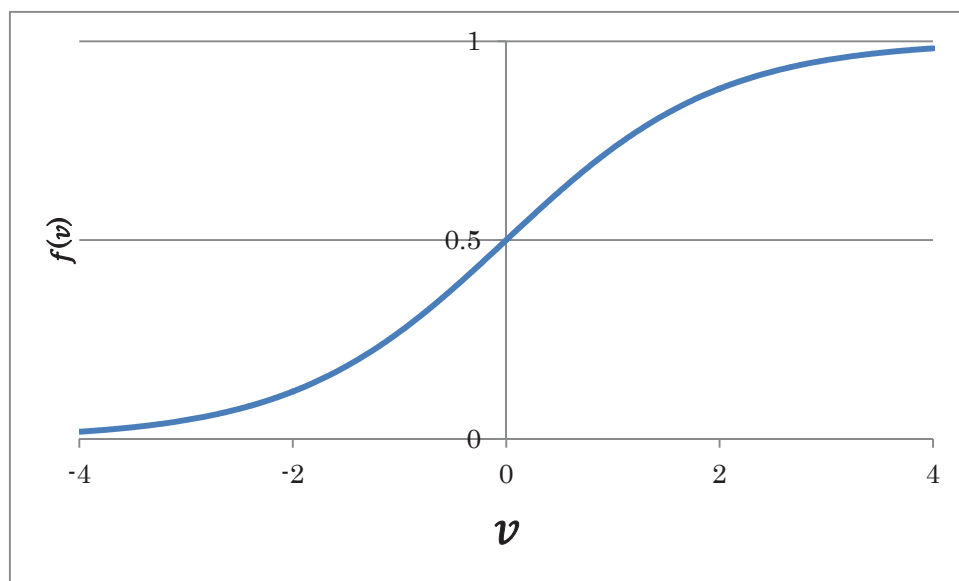



図 4-3 シグモイド関数

4-2 学習アルゴリズム

各層間の各ユニット間に与えられているが結合強度をまとめてwとする．4-1 章にて述べた「教師データをネットワークに次々に提示する」ということは，入力データを実際にネットワークに入力し，その時のネットワークの出力が教師データの出力とどの程度違っているのかをネットワークに教えることである．サンプルデータがM個あるとする．ネットワークの重み係数が w のとき，s 番目のサンプルデータをネットワークに入力したときの出力を

$$y_s^w = (y_{s1}^w, y_{s2}^w, \dots, y_{sL}^w)$$

とし(出力層にはL個のユニットがあるとする)，これが教師データの出力(目標出力)

$$y_s^d = (y_{s1}^d, y_{s2}^d, \dots, y_{sL}^d)$$

とどれくらい違っているかを考える．食い違いを評価する方法はいろいろ考えられるが，その中で最も簡単なのが，二つの値の差の 2 乗を計算する手法である．例えば出力層の 1 番目のユニットについて，実際の出力とお手本の出力の差の 2 乗である$(y_{s1}^w - y_{s1}^d)^2$を計算すると，この値が大きいほど両者の食い違いが大きく，0 に近いほど食い違いが小さいと評価できる．これをすべての出力ユニットについて計算し足し合わせたもの

$$E(w) = \frac{1}{2}\sum_{i=1}^{L}(y_{si}^w - y_{si}^d)^2$$

を出力誤差の評価量(評価関数または誤差関数)として用いることができる．学習は，評価関数$E(w)$なるべく小さく，できれば 0 にする方向に結合強度wを調整していくことになる．結合強度wに対して評価関数$E(w)$は一般に図 4-4 のような山あり谷ありの曲面になっている．

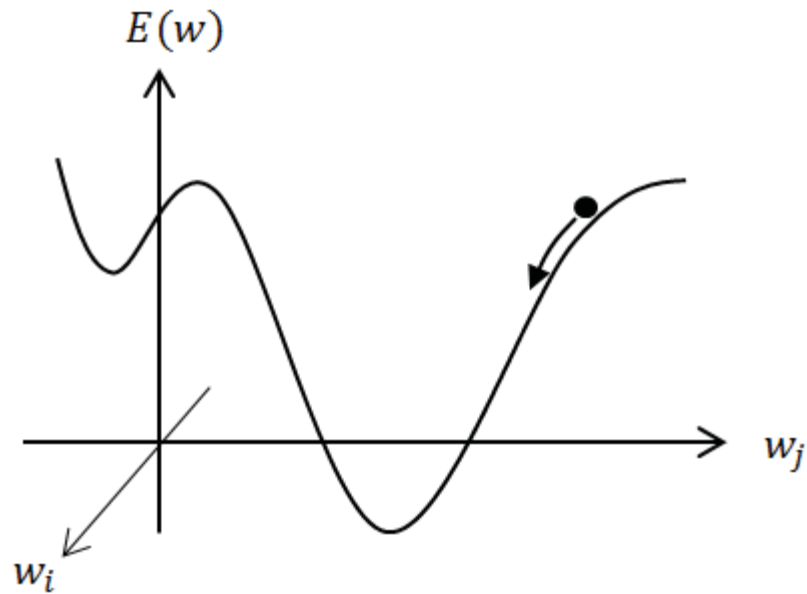



図 4-4 評価関数の曲面

そこで，最初wを適当に(ランダムに)決めて，そこからこの曲面を谷へ降りてくようにしてwを変えていけば$E(w)$はどんどん小さくなっていくはずである．具体的には次のようにwを変えていく．

$$w_{ij}^k \leftarrow w_{ij}^k - \varepsilon \frac{\partial E}{\partial w_{ij}^k}$$

ここで，εは小さな正の定数である．これは，最急降下法とも呼ばれ，曲面上をもっとも急な傾斜方向に降りていく方法として知られている．問題は$\partial E/\partial w_{ij}^k$をどのようにして求めるかである．
まず，各ユニットに対して次のような変数

$$p_i^k = \frac{\partial E}{\partial v_i^k}$$

を定義すると，

$$\frac{\partial E}{\partial w_{ij}^k} = p_i^{k+1} u_j^k$$

が成り立つので，信号pが求まれば問題の$\partial E/\partial w_{ij}^k$が求まることになる．ここで，もう一度

チェーンルールを用いると，

$$\frac{\partial E}{\partial v_i^k} = \sum_j \frac{\partial E}{\partial v_i^{k+1}} f_k'(v_i^k)\, w_{ij}^k$$

すなわち

$$p_i^k = \sum_j p_j^{k+1} f_k'(v_i^k)\, w_{ij}^k \qquad (1)$$

とかけることがわかる．これは，ある層における信号pが求まれば，これを用いてその一つ手前(入力層に近い側)の層における信号pが計算できることを意味している．出力層においてこの信号は，

$$p_i^N = f'\left(v_i^N\right)\left(y_{si}^w - y_{si}^d\right) \quad (N\text{は層の数})$$

と書くことができ，これを出力層から流し込んでやれば，式(1)に従って信号pが出力層から入力層の方向へ逆向きに次々と計算できる．

このように誤差信号が逆向きに伝わるため，このアルゴリズムはバックプロパゲーション(誤差逆伝播法)と呼ばれる．

5章 アプリケーションの作成

5-1 アプリケーションの構成

前章までに整備した環境を用いて，ロボットアームを操作するアプリケーションを構築する．スマートフォンを持った腕を動かすことで，ロボットアームを人間の腕と同じように動作させることを目標とする．そのためにスマートフォンで取得できるセンサの値を加工したものを Bluetooth で PandaBoard に送信し，PandaBoard からシリアル通信を用いてモータを制御する(図 5-1)．シリアル通信の規格として今回は RS-485 を用いる．RS-U485 は USB を RS-485 に変換する変換コネクタである．

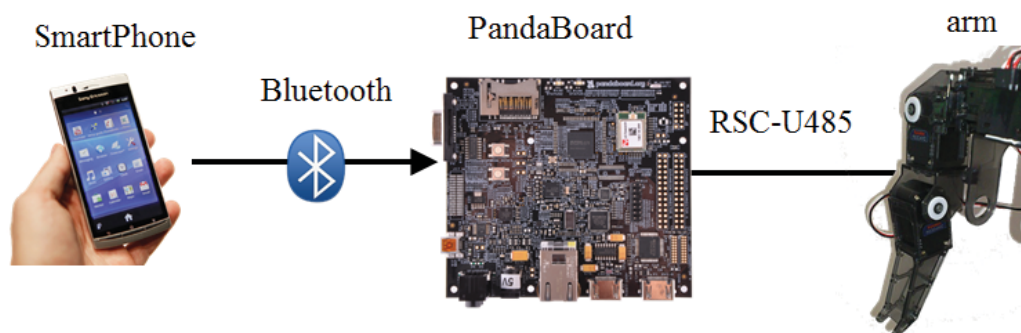



図 5-1 アプリケーションの構成

以下にロボットアームの構成について記す．

ロボットアームの部品は HPI 製 GR-001 の一部を利用した．使用したパーツリストを表 5-1 にまとめる．サーボモータは全部で 6 個使用し可動部は図 5-2 のようになっている．

表 5-1 ロボットアームパーツリスト

部品リスト	製品名
サーボモータ	双葉電子工業製　RS304MD（6 個）
肩	G-ROBOTS GR-001　肩(L/R)
上腕	G-ROBOTS GR-001　上腕(L/R)
ひじ	G-ROBOTS GR-001　ひじ(L/R)
手	G-ROBOTS GR-001　手(L/R)
サーボカバー	G-ROBOTS GR-001　サーボカバー(6 個)
ボディ	G-ROBOTS GR-001　ボディ
POM ワッシャー	POM ワッシャー　2×12×1mm(4 個)
ラバーブッシュ	ラバーブッシュ(4 個)
ワッシャー	ワッシャー　M2×6mm(4 個)
プラブッシュ	プラブッシュ(4 個)

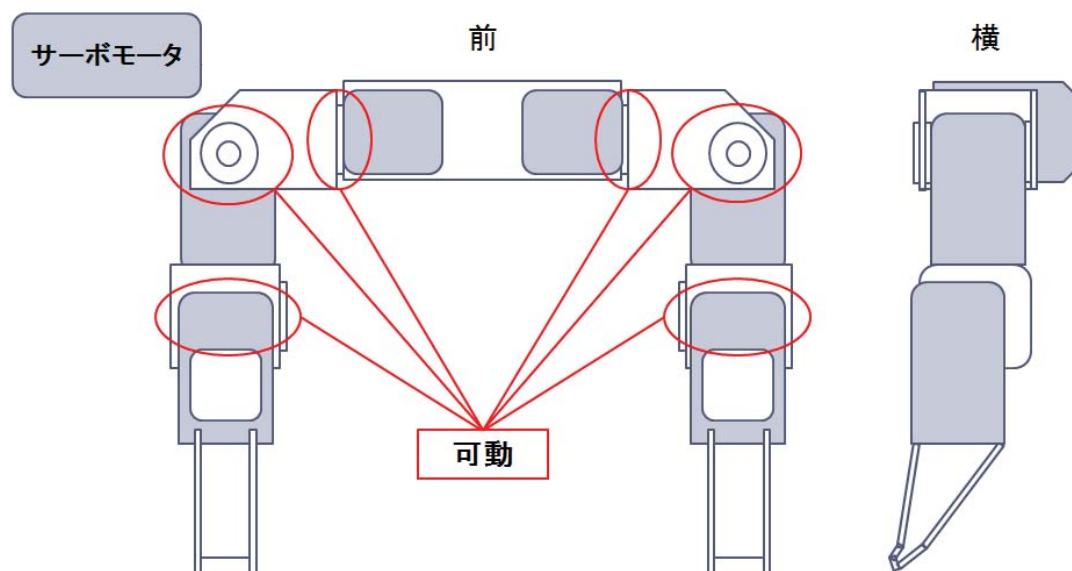



図 5-2 ロボットアームの仕様

5-2 センサ値取得アプリケーション

ロボットアームを人間の腕と同じ動作をさせるためには，センサの値で腕の動きが判別できなければならない．今回利用するセンサは 3 方向加速度センサと傾きセンサである．これらのセンサの値から，腕の動作を判別できるかを確認するため，センサ値を取得するアプリケーションを作成した．

図 5-3 がスマートフォンのセンサ値を取得するアプリケーションである．画面中央にある ON/OFF(トグルボタン)が ON の間，センサのデータを CSV ファイルとして SD カードに保存し続ける．センサの定義，利用法については，5-2-1，5-2-2 章に詳細を示す．

図 5-3 センサ値取得アプリケーション

本研究では動作パターン 2 つに焦点を当て作業を進める．1 つ目は，腕を下に降ろした状態から前方へ約 90 度振り上げ，元の位置まで戻す動作(図 5-4，以下「前振り」)である．2 つ目は，腕を下に降ろした状態から横に 90 度振り上げ，元の位置まで戻す動作(図 5-5，以下「横振り」)である．

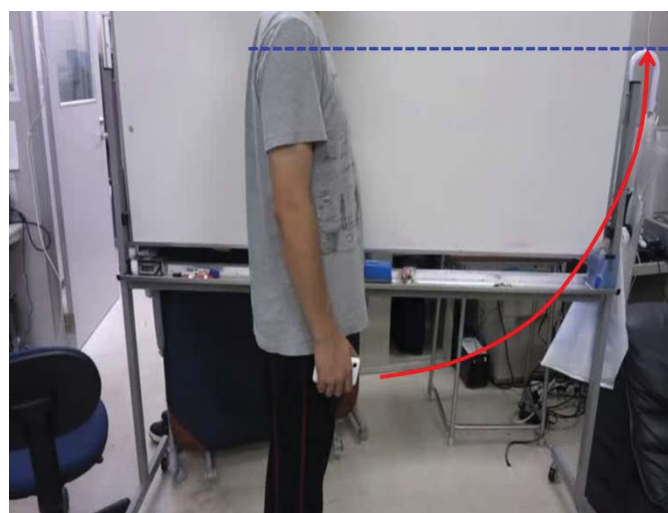



図 5-4 前振り

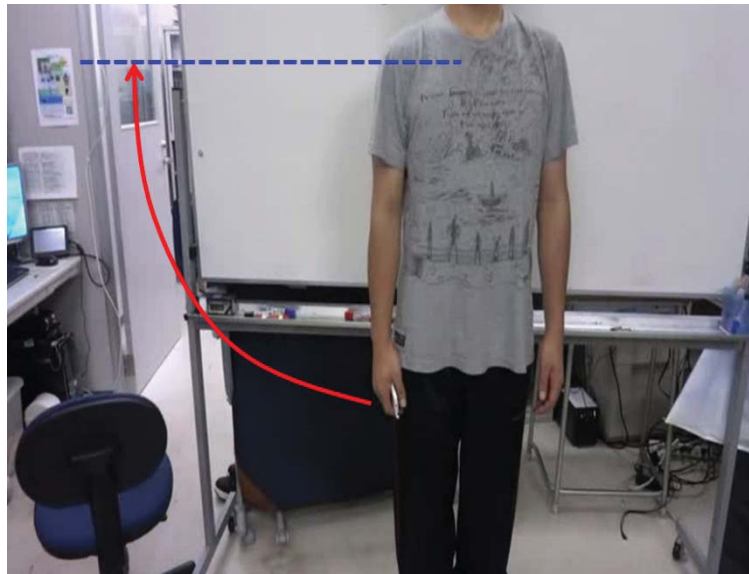



図 5-5 横振り

前振り，横振りそれぞれでセンサの値を取得した．それぞれの動きに対して前振りにのみ対応する値，横振りのみに反応する値がないかを検討した．

5-2-1 センサ値の定義

本研究で利用したセンサは 3 方向加速度センサと傾き(方向)センサである．
以下にそれぞれのセンサ値の定義を記す．

5-2-1-1 加速度センサ

Android の加速度センサではa_x，a_y，a_zの 3 軸の加速度が取得できる．a_x軸は端末の右側から，a_y軸は端末の上部から，a_z軸は端末の正面から受ける加速度の値を取得できる(図 5-6)．なお，a_x，a_y，a_zには重力加速度成分g_x，g_y，g_zと動加速度成分l_x，l_y，l_zが含まれており，

$$a_x = g_x + l_x$$
$$a_y = g_y + l_y$$
$$a_z = g_z + l_z$$

の関係がある．

図 5-6 加速度センサの軸

5-2-1-2 傾きセンサ

傾きセンサは方位角(Azimuth), 傾斜角(Pitch), 回転角(Roll)の 3 つの角度が取得できる．それぞれの値の定義を以下に記す．

・方位角(Azimuth)

方位角は文字通り方位を示すもので北が 0 度，東が 90 度，南が 180 度，西が 270 度を指す(図 5-7).

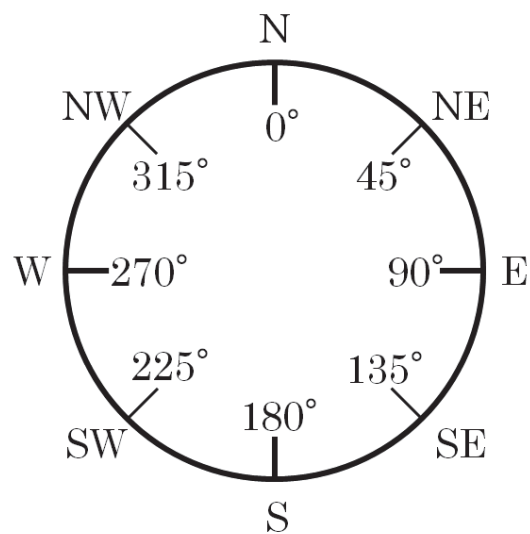



図 5-7 方位角(Azimuth)

・傾斜角(Pitch)

傾斜角は端末を水平に置いた時を 0 度とした上下の傾きであり，-180 度から 180 度の範囲で角度を取得する(図 5-8).

図 5-8 傾斜角(Pitch)

・回転角(Roll)

回転角は端末を水平に置いたときの左右の傾きである．-90 度から 90 度の範囲で角度を取得し，端末を左に傾けるとプラス，右に傾けるとマイナスとなる(図 5-9)．

図 5-9 回転角(Roll)

5-2-2 センサの利用

加速度，傾きセンサを Android で呼び出す方法を記載する.

・加速度センサの値を呼び出すソースは以下の通りである.

加速度センサの値はイベントハンドラの onSensorChanged 内部で TYPE_ACCELEROMETER を用いて呼び出すことができる.

```
switch(e.sensor.getType()){
    case Sensor.TYPE_ACCELEROMETER: {
        TextView x = (TextView)findViewById(R.id.ax);
        x.setText("x:"+String.valueOf(e.values[SensorManager.DATA_X]));

        TextView y = (TextView)findViewById(R.id.ay);
        y.setText("y:"+String.valueOf(e.values[SensorManager.DATA_Y]));

        TextView z = (TextView)findViewById(R.id.az);
        z.setText("z:"+String.valueOf(e.values[SensorManager.DATA_Z]));

        avalue[0] = e.values[SensorManager.DATA_X];
        avalue[1] = e.values[SensorManager.DATA_Y];
        avalue[2] = e.values[SensorManager.DATA_Z];
        break;
    }
}
```

このコードにより avalue[0]に値a_x，avalue[1]に値a_y ，avalue[2]に値a_zが格納される.

・傾きセンサの値の呼び出すソースは以下の通りである. 傾きセンサの値は onSensorChanged 内で TYPE_ORIENTATION を用いて呼び出す.

```
case Sensor.TYPE_ORIENTATION: {
    TextView x = (TextView)findViewById(R.id.ox);
    x.setText("Azimuth:"+String.valueOf(e.values[SensorManager.DATA_X]));

    TextView y = (TextView)findViewById(R.id.oy);
    y.setText("Pitch:"+String.valueOf(e.values[SensorManager.DATA_Y]));

    TextView z = (TextView)findViewById(R.id.oz);
    z.setText("Roll:"+String.valueOf(e.values[SensorManager.DATA_Z]));

    ovalue[0] = e.values[SensorManager.DATA_X];
    ovalue[1] = e.values[SensorManager.DATA_Y];
    ovalue[2] = e.values[SensorManager.DATA_Z];

    break;
}
```

このコードにより ovalue[0]に値 Azimuth，ovalue[1]に値 Pitch，ovalue[2]に値 Roll が格納される.

android バージョン 2.3 以降ではa_x，a_y，a_zから重力成分のみを抜き出した TYPE_GRAVITY が利用できる. これは Android システム内で TYPE_ACCELEROMETER の値にローパスフィルタをかけることで重力成分のみが取得できるようにしたものである. 重力成分の値の取得方法は以下の通りである.

```
case Sensor.TYPE_GRAVITY: {
    TextView x = (TextView)findViewById(R.id.gx);
    x.setText("grx:"+String.valueOf(e.values[SensorManager.DATA_X]));

    TextView y = (TextView)findViewById(R.id.gy);
    y.setText("gry:"+String.valueOf(e.values[SensorManager.DATA_Y]));

    TextView z = (TextView)findViewById(R.id.gz);
    z.setText("grz:"+String.valueOf(e.values[SensorManager.DATA_Z]));

    gvalue[0] = e.values[SensorManager.DATA_X];
    gvalue[1] = e.values[SensorManager.DATA_Y];
    gvalue[2] = e.values[SensorManager.DATA_Z];
    break;
}
```

このコードにより gvalue[0]に値g_x，gvalue[1]に値g_y，gvalue[2]に値g_zが格納される. g_x，g_y，g_zはそれぞれa_x，a_y，a_zに対応した重力加速度の値を取得している.

5-3 取得したセンサ値の比較

取得したセンサの値を比較し，前振りのみ，横振りのみに対応した値があるかを確認する．

5-3-1 加速度センサの取得値

前振り時の加速度(図 5-10)と横振り時の加速度センサa_x，a_y，a_z (図 5-11)の値を比較する．

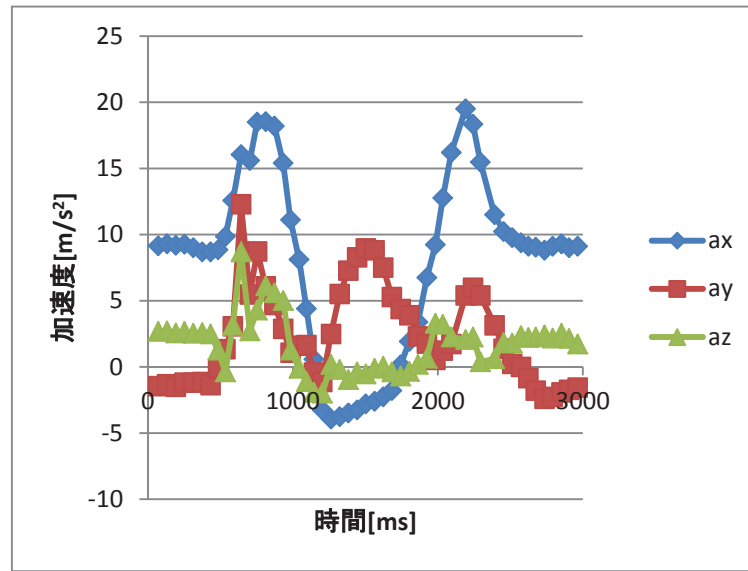



図 5-10 前振り時加速度センサ値

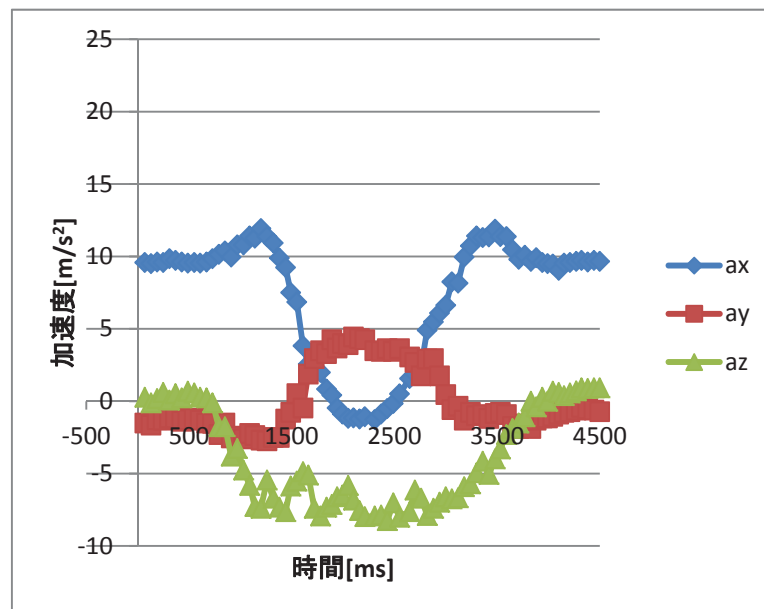



図 5-11 横振り時加速度センサ値

a_x，a_y，a_z全てが前振り，横振りに反応していることがわかる．

さらに重力成分g_x，g_y，g_zのみを抜き出した TYPE_GRAVITY を利用した．センサの値は以下の通りである．(前振り(図 5-12)，横振り(図 5-13))

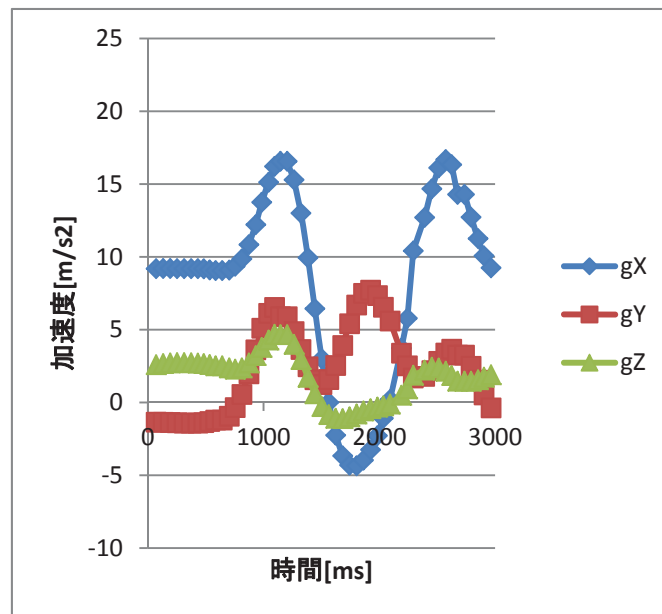



図 5-12 前振り加速度値(重力成分のみ)

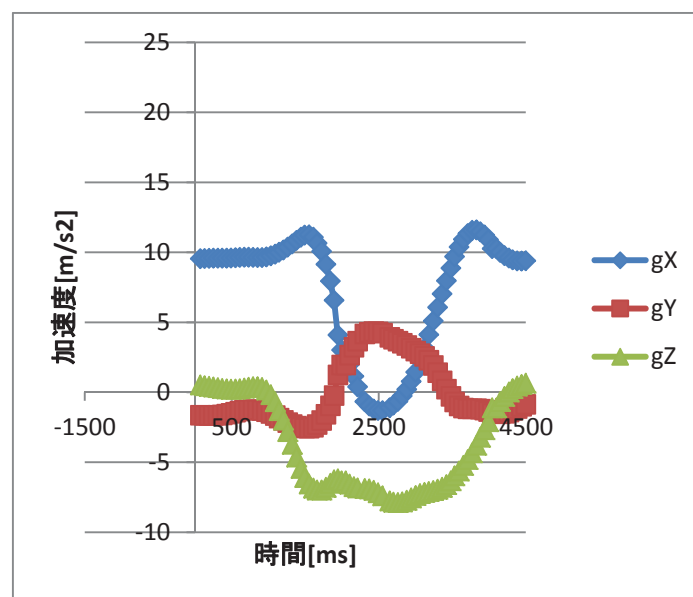



図 5-13 横振り加速度値(重力成分のみ)

値の変化が滑らかになったが，a_x，a_y，a_z同様，前振り，横振りどちらにも値が反応してしまっている．そのため，加速度センサの利用だけでは腕の動作を判別することはで難しいことがわかる．

5-3-2 傾きセンサの取得値

次に，前振り時(図 5-14)と横振り時（図 5-15)の傾きセンサの値を比較する．

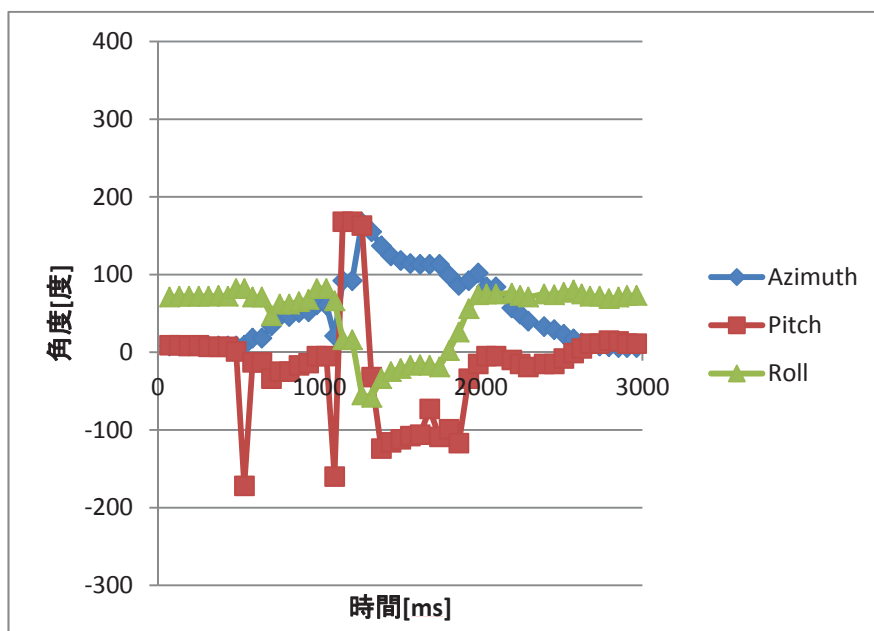



図 5-14 前振り時傾きセンサ値

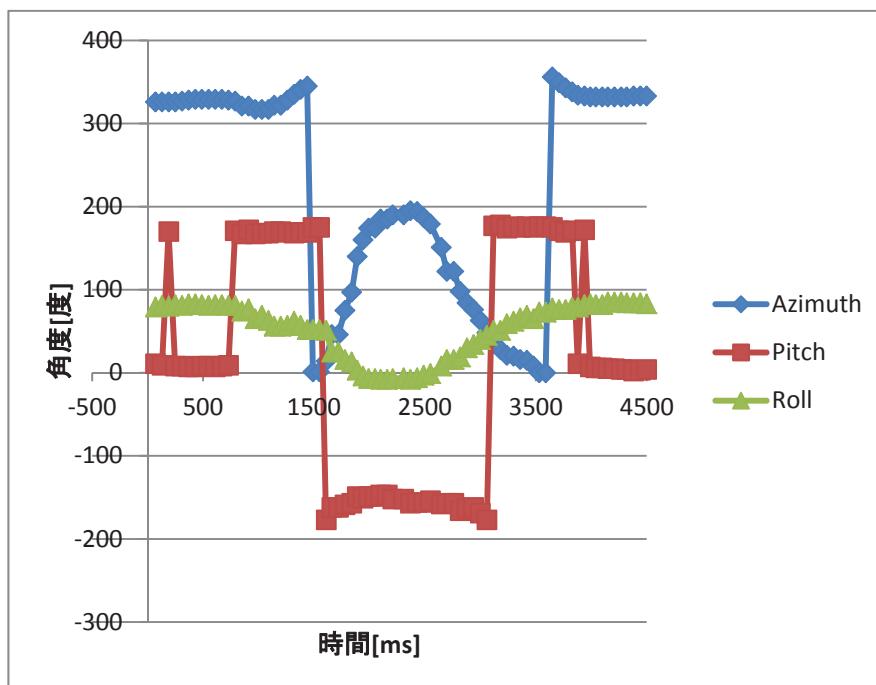



図 5-15 横振り時傾きセンサ値

Azimuth，Pitch，の値が大きく変化している部分が確認できる．これはセンサの取得値に以下のような境界点があるため発生する．

Azimuth の境界点
0 から 360 にとぶ：360 から 0 にとぶ

Pitch の境界点
0 から 180 にとぶ：180 から 0 にとぶ
−180 から 180 にとぶ：180 から−180 にとぶ
−180 から 0 にとぶ：0 から−180 にとぶ

これらの境界点付近では値が不連続に飛ぶため大きな値の変化が生じてしまった．このままの値ではモータの制御に利用できない為 ， 値の加工を行う必要がある．

ここで Azimuth は方位角であり，ユーザーの向きにより値が変わる為今回は利用しない．

Pitch の値をとびがないように加工する為，以下の式を導入し，加工した Pitch の値を Pitch2 とする．図 5-16 は前振り時，図 5-17 は横振り時の傾きセンサの加工値である．

```
if(|Pitch| > 90){
  Return 180 - |Pitch|
}else{
  Return |Roll|
}
```

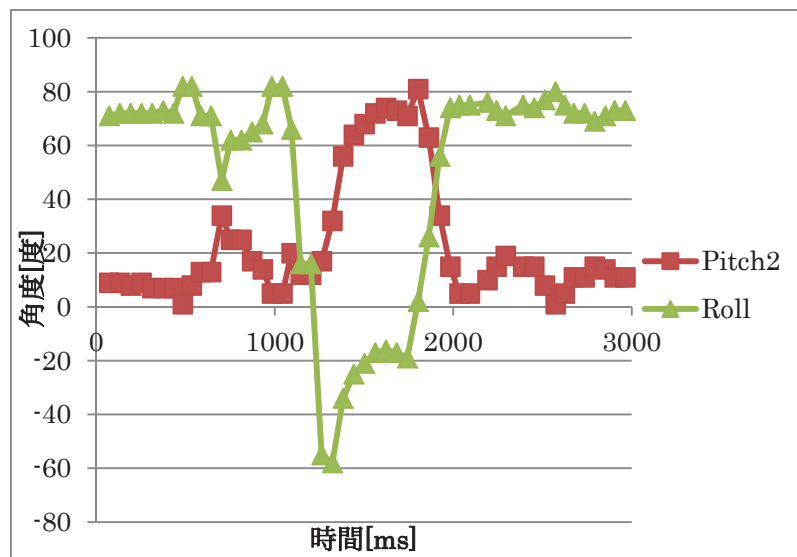



図 5-16 図 5-14 のデータを加工後の前振り傾きセンサ値

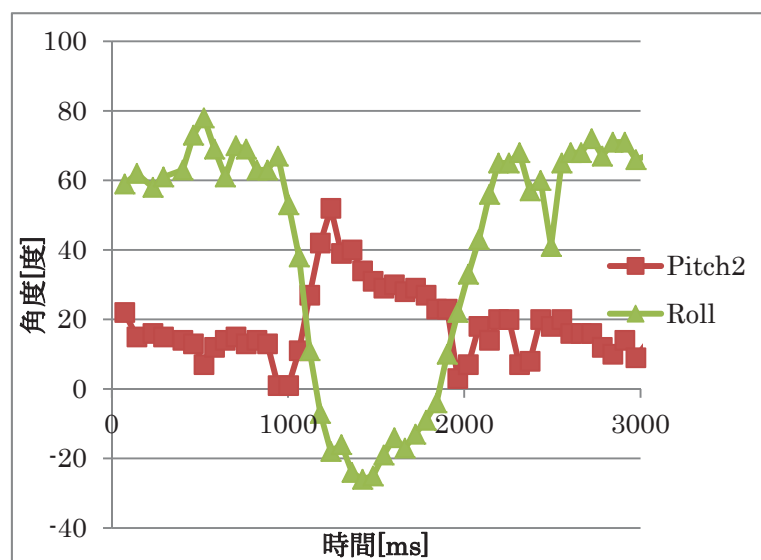



図 5-17 図 5-15 のデータを加工後の横振り傾きセンサ値

加工後の傾きセンサを比較したがセンサの値は全て変動しており，前振りのみ，横振りのみに反応する値は確認できない．

以上のことから加速度センサ，傾きセンサの値をそのまま利用することはできない．そこでバックプロパゲーション(ニューラルネットワーク)による加工を行う．

5-4 バックプロパゲーションによる加工

これまで取得してきたセンサの値を更に加工しモータの制御に利用するため，バックプロパゲーション(ニューラルネットワーク)を利用した．今回は取得しているセンサの値(加速度 3 成分，重力加速度 3 成分，動加速度 3 成分，傾き 3 成分)12 個の中から，加速度 3 成分，重力加速度 3 成分，傾き 2 成分(Pitch と Roll)の自由度を入力とし，前振りにのみ反応する値，横振りにのみに反応する値の 2 つの値を出力として得る．図 5-18 はバックプロパゲーションで値を加工する工程である．ここで，傾き成分ox, oy, ozは，それぞれ Azimuth，Pitch，Roll に対応し，$\widehat{oy}$は Pitch2 に対応している．尚，ネットワークへの入力データは図 5-10，5-11，5-12，5-13，5-16，5-17 に示されている．

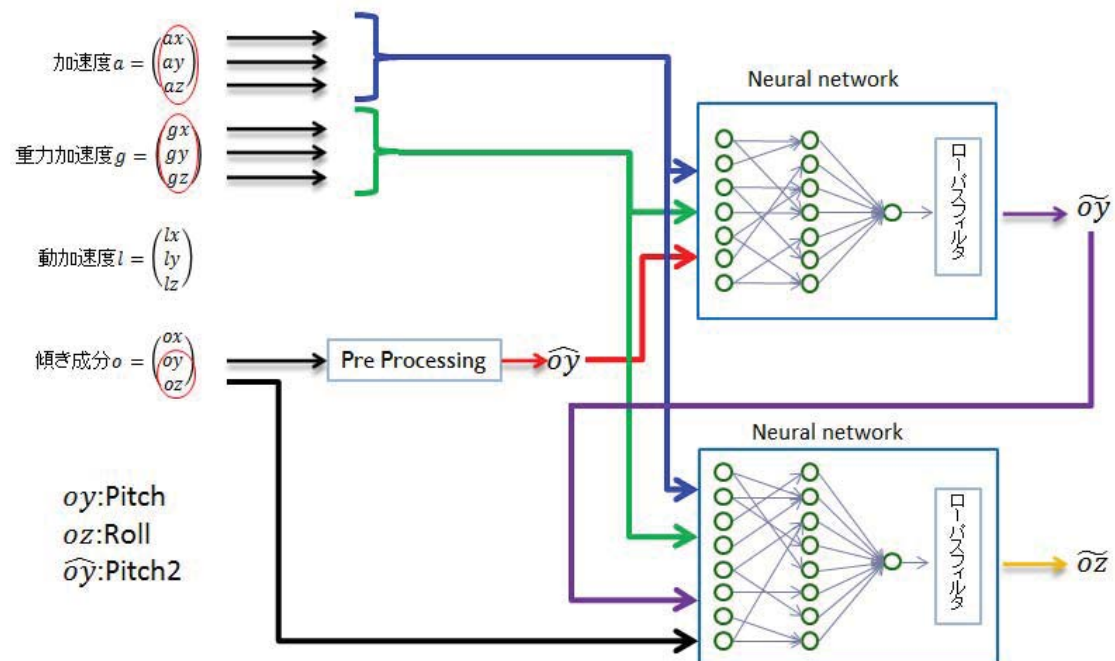



図 5-18 バックプロパゲーションによるセンサ値の加工

バックプロパゲーションの加工を行う際の出力は前振りが図 5-19，横振りが図 5-20 のような波形が出力されるのを理想とする．このデータに値を近づけるための教師データは手作業で作成した．

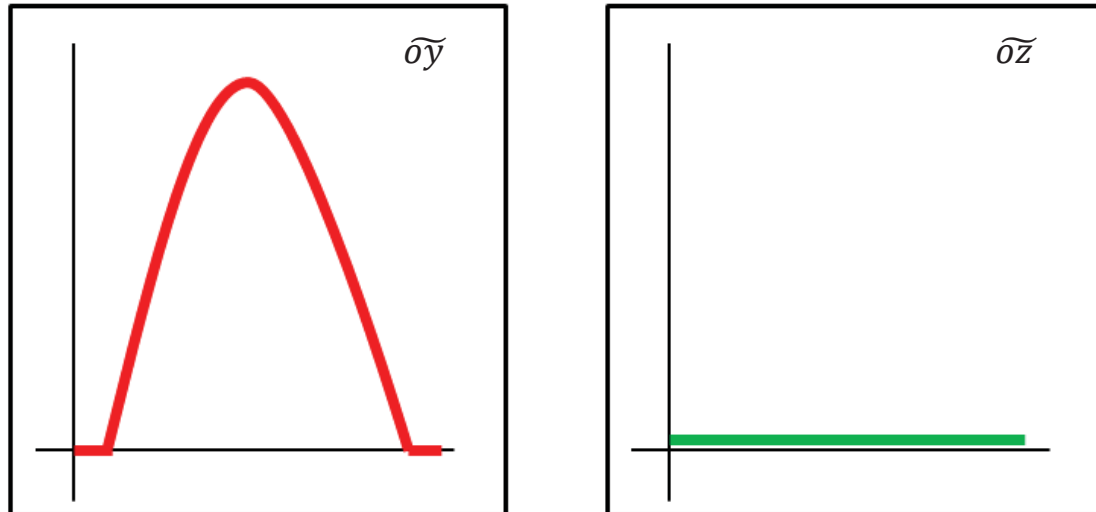



図 5-19 前振りの理想取得値

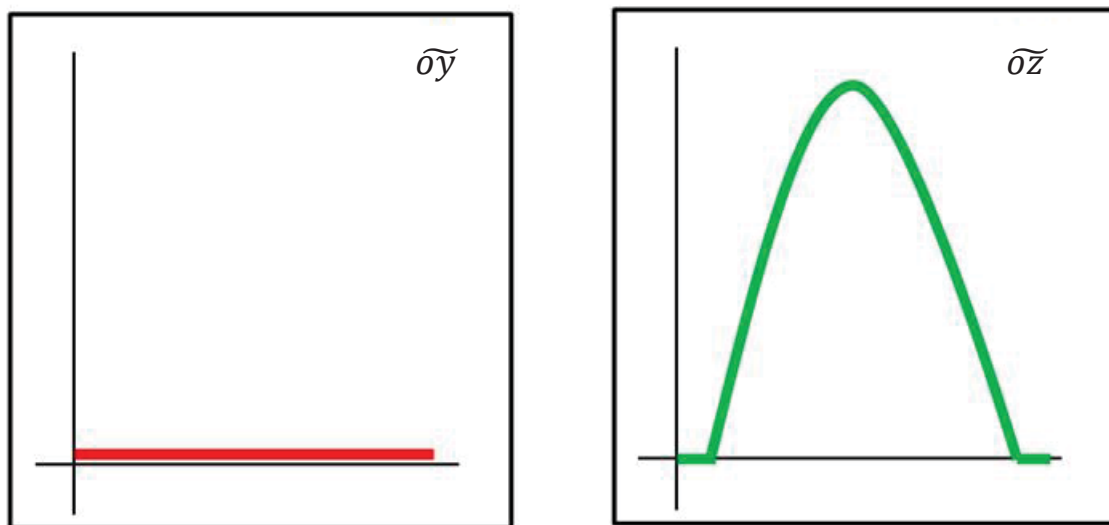



図 5-20 横振りの理想取得値

以上の工程により図 5-16(前振り)と図 5-17(横振り)の値の加工を行った．加工した結果は前振りが図 5-21，横振りが図 5-22 のようになっている．

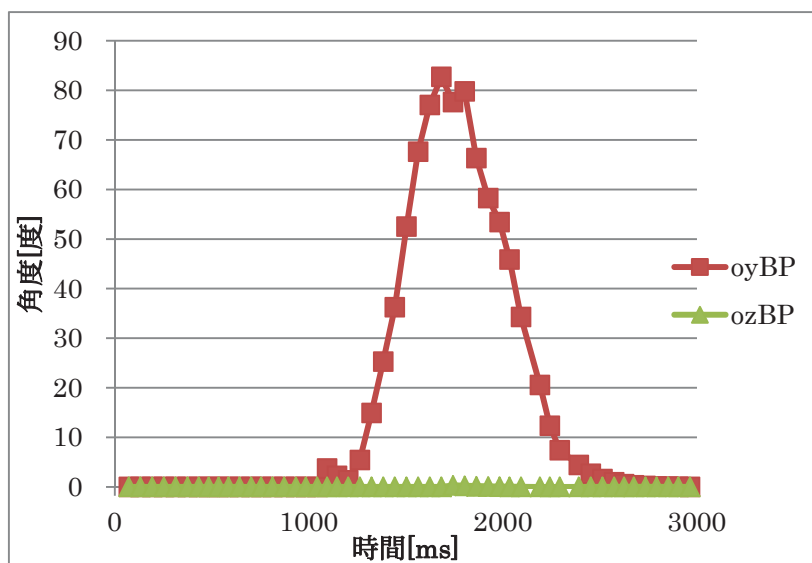



図 5-21 バックプロパゲーション加工による取得値(前振り)

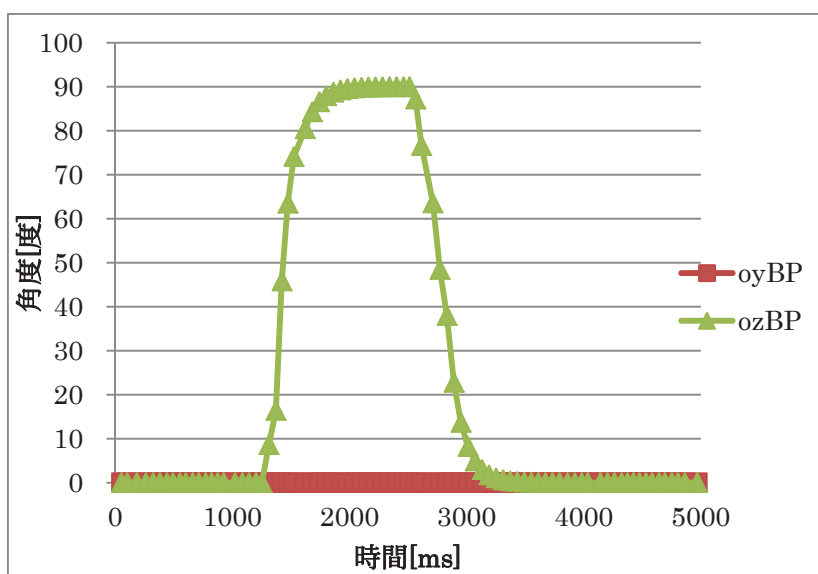



図 5-22 バックプロパゲーション加工による取得値(横振り)

バックプロパゲーションでの値の加工により，前振りだけに反応する値，横振りだけに反応する値の取得ができた．これの値を用いて，Android 搭載のスマートフォン，PandaBoard を利用したモータ制御を行う．

5-5 BluetoothSender

Bluetooth Sender は 5-4 章で解説した加工を行い，$\widetilde{\sigma y}$と$\widetilde{\sigma z}$とモータのサーボ ID を Bluetooth で PandaBoard 側アプリケーションに値を送信するアプリケーションである．本アプリケーションは，左右両方の手に対応させ，メニューボタンにて変更ができる．また，本研究の目的であるロボットアームを腕と同じ動きをさせる「大振り版」とは別に，スマートフォンを水平に持ち，手を前後左右に傾けるだけで操作する「小振り版」のアプリケーションも組み込んだ．「小振り版」はセンサの値の大きなとびがないので，センサ値を線形に加工することでモータを制御できる．「大振り版」，「小振り版」の切り替えは画面上にあるタブにて切り替えができる(図 5-23)．それぞれのモードについて以下で解説する．

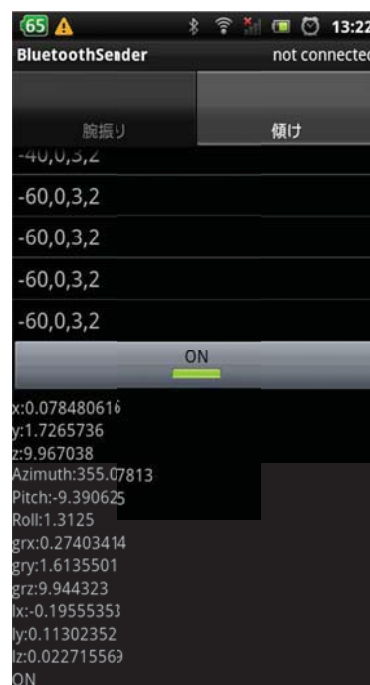



図 5-23 BluetoothSender

5-5-1 大振り(腕振り)版アプリケーション

大振り版アプリケーションは，これまで解説してきたように，スマートフォンを持った腕と同じ動作をロボットアームに行わせるためのアプリケーションである． 5-4 章にて解説したバックプロパゲーションによる加工量$\widetilde{\sigma y}$と$\widetilde{\sigma z}$とモータのサーボ ID を Bluetooth で送信している．動作は左手，右手に切り替えることができ，両腕でロボットアームを動作させることが可能であれる．

5-5-2 小振り(傾け)版アプリケーション

小振り版アプリケーションは，スマートフォンを水平に持った手を前後左右に傾けることでモータを操作する．こちらは傾きセンサの値を線形に加工して利用している．センサの値とモータ角の対応は表 5-2，表 5-3 の通りである．こちらも動作は左手，右手に切り替えることができ，両腕でロボットアームを動作させせることが可能であれる．

表 5-2 小振り版右手用モータ操作値

右手用	傾け値	動作角度
前	Pitch 20 度	肩アーム後方に 90 度
後	Pitch −20 度	肩アーム前方に 90 度
右	Roll −30 度	肘アーム右横に 90 度
左		動作なし

表 5-3 小振り版左手用モータ操作値

左手用	傾け値	動作角度
前	Pitch 20 度	肩アーム後方に 90 度
後	Pitch −20 度	肩アーム前方に 90 度
右		動作なし
左	Roll 30	肘アーム左横に 90 度

5-6 BluetoothReceiver

BluetoothReceiver は BluetoothSender から送られた値を基にモータを制御するための組み込みボード向けアプリケーションである(図 5-24).

ReceiverRighthand connected:SO-01C

SO-01C: -160,0,3,2
SO-01C: -160,0,3,2
SO-01C: -160,0,3,2
SO-01C: -160,0,3,2
SO-01C: -160,0,3,2
SO-01C: -160,0,3,2
SO-01C: -180,0,3,2
SO-01C: -160,0,3,2

0:07

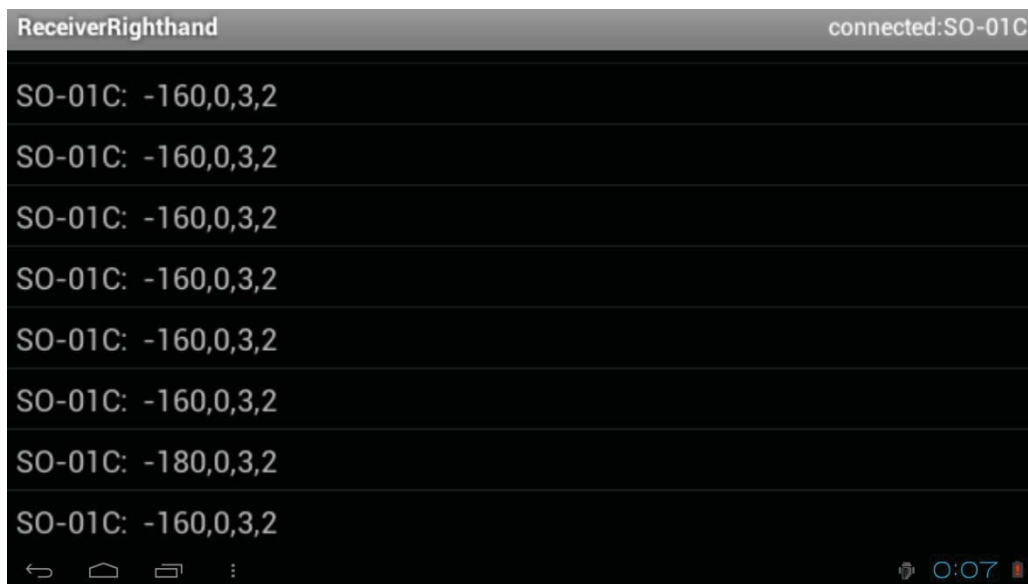



図 5-24 Bluetooth Receiver 画面

こちらは起動するだけで特に操作は行わない．アプリケーションを起動させるとモータのトルクが ON になり，アームは初期位置(値 0)まで戻る(図 5-25).

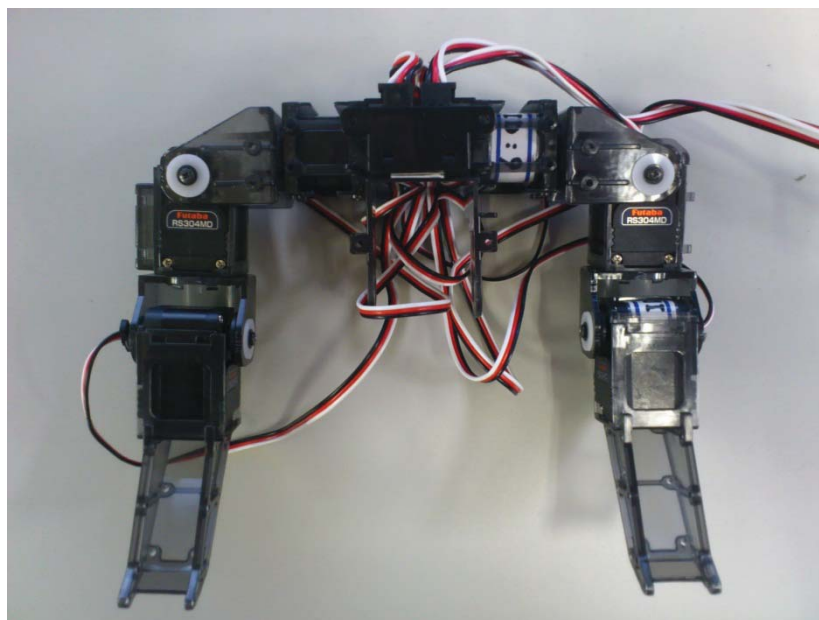



図 5-25 ロボットアーム初期位置

起動時にモータのトルクを ON にし，モータ位置を 0 にリセットするためのソースは下記の通りである．

まず，以下はトルクの ON/OFF をパケットで送信する関数 TorqueOnOff の一部である．Servo_id を 255 にしてモータすべてのトルクを変更している．

```
void TorqueOnOff(int sMode){
	byte sum;
	Servo_id = 255;
	sendbuf[0]  = (byte)0xFA;
	sendbuf[1]  = (byte)0xAF;
	sendbuf[2]  = (byte)Servo_id;
	sendbuf[3]  = (byte)0x00;
	sendbuf[4]  = (byte)0x24;
	sendbuf[5]  = (byte)0x01;
	sendbuf[6]  = (byte)0x01;
	sendbuf[7]  = (byte)(sMode&0x00FF);
	// チェックサムの計算
	sum = sendbuf[2];
	for(int i = 3; i < 8; i++ ){
		sum = (byte)(sum ^ sendbuf[i]);
	}
	sendbuf[8] = sum;

	writeData(sendbuf, 9);
}
```

ここで sMode が 1 の場合モータのトルクが ON になり，0 の場合モータのトルクが OFF になる．

以下は Android アプリ起動時に呼び出される Activity，onCreate の一部である．

```
public void onCreate(Bundle savedInstanceState) {

//～～～～～～～～～～～～～～～～～～中略～～～～～～～～～～～～～～～～～～～

	TorqueOnOff(1);
	try{
		Thread.sleep(200);
	}catch(Exception e){}

	RSMove(255,0,0);
}
```

TorqueOnOff(1)にてトルクを ON にし，RSMove(255,0,0)でサーボ ID(255)，モータの指定角(0 度)，モータの指定角到達までの速度(最速)を指定している．

スマートフォン側から受信するデータは値を「,」で区切った一つの文字列として送られてくる．この値を下記のコードによって一つ一つの値に分解している．受信するデータはスマートフォン側で加工された 2 つの値$\widetilde{oy}$と$\widetilde{oz}$である．

```
public void processData(String msg){
	if(D) Log.e(TAG, "processData entered");

	try{
		st = new StringTokenizer(msg,",");
	}catch(Exception e){
		return;
	}
		try{
			oy = Integer.valueOf(st.nextToken()).intValue();
		}catch(Exception e){
			return;
		}
		try{
			oz = Integer.valueOf(st.nextToken()).intValue();
		}
		catch(Exception e){
			return;
		}
}
```

ここで `st = new StringTokenizer(msg,",");` は「,」で文字列を区切るための StringTokenizer のオブジェクトを生成しており, `st.nextToken();` で文字列に「,」を発見したら，そこまでの文字列が区切られる．

受信する値はモータの可動範囲-1500～1500(1 度=10)である．

5-7 スマートフォンと PandaBoard の接続，モータ制御

BluetoothSender と BluetoothReceiver を用いてモータの制御を行う．端末接続，アームの制御の手順は以下の通りである．この際，事前にスマートフォンと PandaBoard の Bluetooth をペアリングしておく必要がある．

- 電源に接続したロボットアームのモータ(RS-304MD)をシリアル変換器を通して PandaBoard に接続する．
- BluetoothReceiver を起動する．このときモータのトルクが ON になりロボットアームは初期位置まで戻る．
- BluetoothSender を起動する．
- BluetoothSender でメニューボタンで右手，左手の選択をする(図 5-26)．

図 5-26 左右の選択

- メニューボタンで「Connect a device」を選択し，ペアリングした PandaBoard 端末と接続する(図 5-27)．

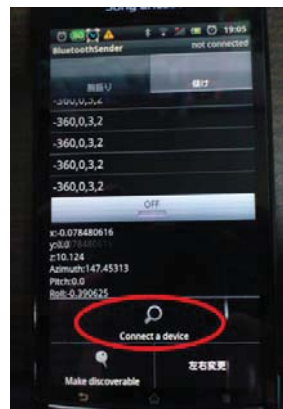



図 5-27 端末同士の接続

・接続が確認できたら，スマートフォンを持った腕を下に降ろし(初期位置)，アプリケーション中央の ON/OFF ボタン(トグルボタン)をタッチする(図 5-28)．正しく接続されていればボタンが ON の間だけ値の送信をし，スマートフォンを持った腕と同じようにロボットのアーム(図 5-29)が動作する．

図 5-28 センサ値の送信

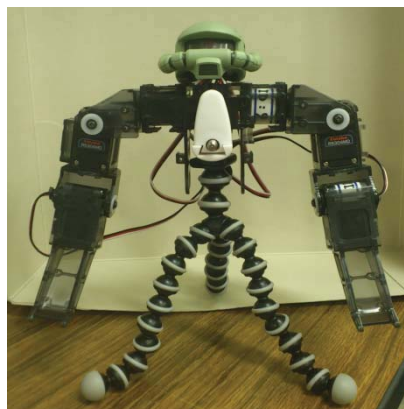



図 5-29 ロボットの全貌

実際にロボットアームの制御を試みたところ，腕の動作には少し遅れる結果にはなったが，ロボットアームは腕と同じ動作をした(図 5-30).

図 5-30 ロボットアームの動作実験

6章　結論

本研究では，スマートフォンのセンサ値を用いたロボットアームの遠隔操作アプリケーションを作成した．スマートフォンの 3 方向加速度センサ，傾きセンサの値を利用することにしたが，センサ値は前振り，横振り動作に全てのセンサの値が反応してしまった．前振りのみ，横振りのみのそれぞれに反応する値を作成するため，バックプロパゲーションによる値の加工を行い，それぞれの動作のみに反応する値を得ることができ，スマートフォンをも持った腕と同じ動作をさせることが可能になった．

しかし，アームの動作は，腕の動きに少し遅れて動作する．これはニューラルネットワークを通した際に発生した時間の誤差である(図 6-1)．図 6-1 を見ると約 100ms の遅れが見られる．

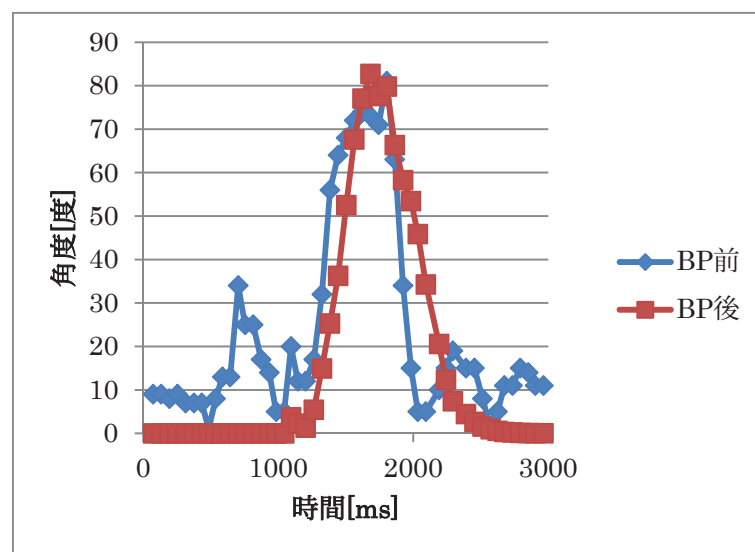



図 6-1 ニューラルネットワーク前後の値比較

ニューラルネットワークに加える変数のより良い組み合わせを見つけることで，アーム動作の誤差を更に減らすことができるのではないかと考えている．

今回の研究では，モータの動作を前振り，横振りの 2 パターンに焦点を当て作業を行ったが，センサによる動きの判別を可能にし，腕の動作にあわせたロボットアームの動作を実現した．

動作パターンの増加やニューラルネットワークの向上により，より自由にロボットアームの制御を行えると考えられる．

図表目次

謝辞

本研究を進めるにあたり，懇切丁寧なご指導をして頂きました金丸隆志准教授，また，本論文の執筆にあたり，有意義なご意見を頂きました疋田光孝教授，濱根洋人准教授に心より厚く御礼申し上げます．

学部生の時に研究室に所属してから 4 年間，多くのことを学ばせて頂きました．同じ失敗を繰り返しご迷惑おかけしてしまうことも多々ありましたが，その度に有意義なご指導をして頂きました． 研究以外の事に対しても IT に関する様々な知識を身につけることができました．改めて金丸准教授に感謝の意を表します．

また，同研究室の後輩達，他研究室の同輩方にも数々の相談に乗って頂き，有意義な助言をして頂きました．

修士 2 年間，同研究室で学んできた高木氏にも心より感謝致します．

その他，私を支えてくださった皆様にも感謝の気持ちを表し，謝辞とさせていただきます．

参考文献

[1] 携帯電話の普及率推移をグラフ化してみる(2011 年版)
http://www.garbagenews.net/archives/1805400.html
[2] Venture Now 2015 年 3 月末にスマホ契約数がガラケーを超える～スマホ市場規模予測
http://www.venturenow.jp/news/2011/07/08/1714_013084.html
[3] smarterous 国内 Android マーケット現況（上）Android 端末の普及台数予測とユーザ属性、国内アプリマーケットまとめ [2011 年 4 月]
http://mottoi.posterous.com/android-market-in-japan-and-promotion
[4] Tech Crunch Android マーケット、60 億ダウンロード間近。
http://jp.techcrunch.com/archives/20110912report-android-market-nearing-6-billion
-downloads-weather-apps-are-makin-it-rain/
[5] IT Pro　第 1 回 Android の構造
http://itpro.nikkeibp.co.jp/article/COLUMN/20091127/341206/
[6] Android Development Kit for Sitara Microprocessors
http://www.ti.com/tool/androidsdk-sitara
[7] android-development-environment
http://code.google.com/p/android-development-environment/
[8]研究室ホームページ
http://brain.cc.kogakuin.ac.jp/research/froyobb.html
[9] バックプロパゲーション
http://ja.wikipedia.org/wiki/%E3%83%90%E3%83%83%E3%82%AF%E3%83%
97%E3%83%AD%E3%83%91%E3%82%B2%E3%83%BC%E3%82%B7%E3%83%A7%
E3%83%B3
[10]C でつくる脳の情報システム　中野 馨＝編著 知能システム研究会 近代科学者
[11] PDF RS303MR／RS304MD 取扱説明書
http://www.futaba.co.jp/robot/command_type_servos/rs304md.html
[12] Using RXTX
http://rxtx.qbang.org/wiki/index.php/Using_RXTX
[13] アンドロイド開発環境の構築(その 5)NDK のインストールと設定
http://www.usefullcode.net/2010/12/android_sdk_inst05.html
[14] Cygwin
http://www.cygwin.com/

付録

I. kernel の変更，Android の環境設定

kernel の変更を行い「USB カメラ」，「無線 LAN」，「タッチパネル」を対応させる．尚，「Bluetooth」と「USB-シリアル変換デバイス」については 2 章にて解説した．vender_tk-beagle-froyo2.2.x-20110725.tar.gz によりすべての変更は適用されているが，ここではその変更点をまとめる．

まずは下記の kernel ファイルの内部を変更することでデバイスの対応を行う．

```
$ANDROID/kernel-beagleboard/arch/arm/configs/sola_omap3_beagle_android_defconfig
```

I-ｉ 各種デバイスの有効化

(a) kernel の変更

#USB Camera

```
CONFIG_VIDEO_DEV=y
CONFIG_VIDEO_V4L2_COMMON=y
CONFIG_VIDEO_V4L1_COMPAT=y
CONFIG_VIDEO_MEDIA=y
CONFIG_MEDIA_TUNER=y
CONFIG_MEDIA_TUNER_SIMPLE=y
CONFIG_MEDIA_TUNER_TDA8290=y
CONFIG_MEDIA_TUNER_TDA9887=y
CONFIG_MEDIA_TUNER_TEA5761=y
CONFIG_MEDIA_TUNER_TEA5767=y
CONFIG_MEDIA_TUNER_MT20XX=y
CONFIG_MEDIA_TUNER_XC2028=y
CONFIG_MEDIA_TUNER_XC5000=y
CONFIG_VIDEO_V4L2=y
CONFIG_VIDEO_CAPTURE_DRIVERS=y
```

#Wireless LAN

```
CONFIG_CFG80211=y
CONFIG_NL80211=y
CONFIG_WIRELESS_EXT=y
CONFIG_WIRELESS_EXT_SYSFS=y
CONFIG_MAC80211=y
CONFIG_MAC80211_RC_PID=y
CONFIG_MAC80211_RC_DEFAULT_PID=y
CONFIG_MAC80211_RC_DEFAULT="pid"
CONFIG_MAC80211_MESH=y
CONFIG_MAC80211_LEDS=y
CONFIG_IEEE80211=y
CONFIG_IEEE80211_CRYPT_WEP=y
CONFIG_IEEE80211_CRYPT_CCMP=y
CONFIG_IEEE80211_CRYPT_TKIP=y

CONFIG_WLAN_80211=y
CONFIG_HOSTAP=y
CONFIG_HOSTAP_FIRMWARE=y
```

```
CONFIG_HOSTAP_FIRMWARE_NVRAM=y
CONFIG_RT2X00=m
CONFIG_RT2X00_LIB=m
CONFIG_RT2X00_LIB_USB=m
CONFIG_RT2X00_LIB_FIRMWARE=y
CONFIG_RT2X00_LIB_LEDS=y
CONFIG_RT73USB=m
CONFIG_RT73USB_LEDS=y
CONFIG_CRC_ITU_T=y
```

#タッチパネル関連

```
# CONFIG_TOUCHSCREEN_USB_EGALAX is not set // 有効だったものを無効にする
CONFIG_TOUCHSCREEN_USB_GENERAL_TOUCH=y
```

以上の変更が完了した後，以下のコマンドにより kernel のビルドを行う．

```
$ cd $ANDROID/kernel-beagleboard
$ make ARCH=arm CROSS_COMPILE=../prebuilt/linux-x86/toolchain
/arm-eabi-4.4.0/bin/arm-eabi-
sola_omap3_beagle_android_defconfig
$ make ARCH=arm CROSS_COMPILE=../prebuilt/linux-x86/toolchain
/arm-eabi-4.4.0/bin/arm-eabi- uImage modules
```

kernel のビルドが終了した後，下記のディレクトリにある uImage をパーティション分けした SD カードの DISK1 入れる．

```
$ANDROID/kernel-beagleboard/arch/arm/boot/uImage
```

次に Android ソースの変更を行う．

(b) USB カメラの利用

$ANDROID/vendor/sola/beagleboard/BoardConfig.mk を編集用に開き、

USE_CAMERA_STUB := true

という行をコメントアウトする(行先頭に#を付ける).

#USE_CAMERA_STUB := true とする．

次に$ANDROID/system/core/init/devices.c の以下の部分を 666 設定(誰でも読み書き可能)に変更することでユーザー利用可にしている．尚，この設定は Android2.3 以降では，/system/etc/uevent.rc 行う．

```
{"/dev/ttyUSB0",    0666, AID_ROOT,  AID_ROOT,  0},
{"/dev/ttyUSB1",    0666, AID_ROOT,  AID_ROOT,  0},
```

(c) 無線 LAN の利用

次に無線 LAN 利用のための Android ソースを変更する.

$ANDROID/vendor/sola/beagleboard/BoardConfig.mk に下記の 2 行を追加する.

```
WPA_BUILD_SUPPLICANT := true
BOARD_WPA_SUPPLICANT_DRIVER := WEXT
```

$ANDROID/vendor/sola/omap3/image/beagleboard/に下記の内容の wpa_supplicant.conf を作成する.

```
Update config=1
Ctrl_interface=wlan0
Eapol_version=1
Ap_scan=1
Fast_reauth=1
```

次に$ANDROID/vendor/sola/omap3/image/beagleboard-image.sh のファイルを編集用に開く.「cp $ANDROID/vendor/sola/omap3/image/beagleboard/rt73.bin ... (略)」の次に下記の 2 コマンドを追加する.

```
mkdir -p $ANDROID/vendor/sola/omap3/image/beagleboard/android
         /system/etc/wifi
cp $ANDROID/vendor/sola/omap3/image/beagleboard/wpa_supplicant.conf $ANDROID/
   vendor/sola/omap3/image/beagleboard/android/system/etc/wifi
```

次に，$ANDROID/vendor/sola/beagleboard/init.omap3.rc を編集する．on boot の次の行に下記の行を追加する.

```
Usetprop wifi.interface "wlan0"
setprop wlan.driver.status "ok"
setprop dalvik.vm.heapsize "32m"
```

同じく$ANDROID/vendor/sola/beagleboard/init.omap3.rc の末尾に下記を追加する.

```
#WIFI
service ifcfg_ralink /system/bin/ifconfig wlan0 up
#  disabled
#  oneshot
#      service      wpa_supplicant      /system/bin/logwrapper
/system/bin/wpa_supplicant       -Dwext       -iwlan0       -c
/system/etc/wifi/wpa_supplicant.conf -dd
#   disabled
#   group wifi
# for Android private socket
service  wpa_supplicant  /system/bin/wpa_supplicant  -dd  -Dwext
-iwlan0 -c /system/etc/wifi/wpa_supplicant.conf
   socket wpa_wlan0 dgram 660 wifi wifi
   group system wifi inet
   disabled
   oneshot
service dhcpcd /system/bin/logwrapper /system/bin/dhcpcd -d wlan0
   disabled
   oneshot
   #group system dhcp
on property:init.svc.wpa_supplicant=stopped
   stop dhcpcd
```

次に、$ANDROID/vendor/sola/beagleboard/system.prop を編集用に開き，末尾に下記 1 行を追加する.

```
wifi.interface = wlan0
```

次に、$ANDROID/vendor/sola/beagleboard/init.rc を編集用に開き，既存の下記 3 行をコメントアウト(行先頭に#)する.

```
mkdir /data/misc/wifi 0770 wifi wifi
chmod 0770 /data/misc/wifi
chmod 0660 /data/misc/wifi/wpa_supplicant.conf
```

そして，その位置に代わりに以下を追加する.

```
chmod 755 /system/etc/dhcpcd
chmod 755 /system/etc/dhcpcd-run-hooks
chmod 755 /system/etc/dhcpcd-hooks
chmod 755 /system/etc/dhcpcd-hooks/*

chown 1010 1010 /system/etc/wifi
chmod 770 /system/etc/wifi
chown 1010 1010 /system/etc/wifi/wpa_supplicant.conf
chmod 660 /system/etc/wifi/wpa_supplicant.conf

mkdir /data/misc/wifi
chown 1010 1010 /data/misc/wifi
chmod 770 /data/misc/wifi
mkdir /data/misc/wifi/sockets
chown 1010 1010 /data/misc/wifi/sockets
chmod 770 /data/misc/wifi/sockets
chown 1010 1010 /data/misc/wifi/wpa_supplicant.conf
chmod 660 /data/misc/wifi/wpa_supplicant.conf

mkdir /data/misc/dhcp
chown 1014 1014 /data/misc/dhcp
chmod 0770 /data/misc/dhcp
```

次に、$ANDROID/frameworks/base/wifi/java/android/net/wifi/WifiStateTracker.java を編集用に開き下記のように tiwlan0 を wlan0 に変更する.

```
//mInterfaceName = SystemProperties.get("wifi.interface", "tiwlan0");
mInterfaceName = SystemProperties.get("wifi.interface", "wlan0");
```

次に$ANDROID/external/wpa_supplicant/driver_wext.h と
$ANDROID/external/wpa_supplicant/driver_wext.c を 2-4-3-2 章にてダウンロードした vender_tk-beagle-froyo2.2.x-20110725.tar.gz の中に含まれているものに置き換える.
以上の変更を行った後，Android を再ビルドし，DISK3 に載せれば無線 LAN を利用することができる.

Ⅰ-ⅱ Android の環境設定

Android ソースの変更について以下に記す.

(e) デフォルトでの日本語フォントインストール

Android 起動時にデフォルトで日本語フォントがインストールされるように設定する.
$ANDROID/frameworks/base/data/fonts/Android.mk を開き，DroidSansJapanese.ttf を以下のように追加する.

```
DroidSans-Bold.ttf      ¥
DroidSansJapanese.ttf   ¥
DroidSerif-Regular.ttf  ¥
```

(f) ディスプレイの常時点灯

次にディスプレイの常時点灯の設定を行う.
$ANDROID/packages/apps/Settings/res/values/arrays.xml （英語環境用）において screen_timeout_entries の項目を以下のように末尾に「Never timeout」を追記する.

```
<string-array name="screen_timeout_entries">
       (中略)
       <item>10 minutes</item>
       <item30 minutes</item>
       <item>Never timeout</item>
</string-array>
```

次に，screen_timeout_values の項目を以下のように末尾に値-1 を追記する.

```
<string-array name="screen_timeout_values" translatable="false">
       (中略)
       <!-- Do not translate. -->
       <item>600000</item>
       <!-- Do not translate. -->
       <item>1800000</item>
       <!-- Do not translate. -->
       <item>-1</item>
</string-array>
```

以上の設定を行うことで，設定のタイムアウト時間の設定に「Never timeout」という項目が現れ，ディスプレイの常時点灯が可能となる．ただし，上記の設定は英語環境の場合に限る．日本語環境では，$ANDROID/packages/apps/Settings/res/values-ja/arrays.xml (日本語環境用)にて英語環境と同じように screen_timeout_entries のの項目の 30 分の下に下記 1 行の追加が必要となる.

```
<item msgid="1781492122915870416">"常時点灯"</item>
```

尚、values/arrays.xml は英語環境、values-ja/arrays.xml 日本語環境でそれぞれ用いられる。本来は values-○○の全ての言語のファイルを変更すべきだが、本研究では英語と日本語のみの設定にとどめた。

(g) 機内モードの設定

次に機内モードの設定を行った．誤って機内モードに設定してしまった場合に復帰できなくなってしまうことがあるため, 本研究では無効化した. 変更方法は Android2.2 の場合, $ANDROID/packages/apps/Settings/src/com/android/settings/WirelessSettings.java　を開き，airplane に関する項目をすべてコメントアウトする．すべてコメントアウトした後も機内モードの項目は残るが誤って触れてしまっても何も起こらない.

II. メモリーマップ

双葉電子工業製モータ RS-304MD のメモリーマップは以下の通りである[11]

(a)変更不可領域のメモリーマップ

変更不可領域のメモリーマップは以下のとおりである

変更不可領域のメモリーマップ

領域	ｱﾄﾞﾚｽ No. 10 進	16 進	初期値	名称	内容	R/W
変更不可領域	00	00H	30H(40H)	Model Number L	モデル番号	R
	01	01H	30H	Model Number H	モデル番号	R
	02	02H	03H	Firmware Version	ファームウェアバージョン	R
	03	03H	--	Reserved	予備	-

( ) RS304MD の場合

Figure II-i 変更不可領域のメモリーマップ

・モデル番号(2 バイト，Hex 表記，Read)

モデル番号(サーボ機種)を表す．RS-304MD では以下の値になる．

Model_Number L = 40H

Model_Number H =30H

・ファームウェアバージョン(1 バイト，Hex 表記，Read)

サーボのファームウェアバージョンを表す．

値は，製造時のバージョン(下記の例では 0x03)によって変わる．

Firmware Version =03H

(b)ROM 領域のメモリーマップ

ROM 領域のメモリーマップは以下のとおりである.

RS303MR／RS304MD メモリーマップ（ROM 領域）

領域	ｱﾄﾞﾚｽ No. 10 進	16 進	初期値	名称	内容		R/W
ROM 領域	04	04H	01H	Servo ID	サーボ ID	*C	RW
	05	05H	00H	Reverse	反転		RW
	06	06H	07H	Baud Rate	通信速度	*C	RW
	07	07H	00H	Return Delay	返信遅延時間	*C	RW
	08	08H	DCH	CW Angle Limit L	右リミット角度		RW
	09	09H	05H	CW Angle Limit H	右リミット角度		RW
	10	0AH	24H	CCW Angle Limit L	左リミット角度		RW
	11	0BH	FAH	CCW Angle Limit H	左リミット角度		RW
	12	0CH	00H	Reserved	予備		-
	13	0DH	00H	Reserved	予備		-
	14	0EH	4DH	Temperature Limit L	温度リミット		R
	15	0FH	00H	Temperature Limit H	温度リミット		R
	16	10H	00H	Reserved	予備		-
	17	11H	00H	Reserved	予備		-
	18	12H	00H	Reserved	予備		-
	19	13H	00H	Reserved	予備		-
	20	14H	00H	Reserved	予備		-
	21	15H	00H	Reserved	予備		-
	22	16H	00H	Torque in Silence	無信号時トルク	*P	RW
	23	17H	C8H	Warm-up Time	準備時間	*P	RW
	24	18H	02H	CW Compliance Margin	コンプライアンスマージン		RW
	25	19H	02H	CCW Compliance Margin	コンプライアンスマージン		RW
	26	1AH	08H	CW Compliance Slope	コンプライアンススロープ		RW
	27	1BH	08H	CCW Compliance Slope	コンプライアンススロープ		RW
	28	1CH	64H(58H)	Punch L	パンチ		RW
	29	1DH	00H(02H)	Punch H	パンチ		RW

() RS304MD の場合

*C はコマンド動作時のみ、*P は PWM 動作時のみで有効なパラメータ

Figure II-ii ROM 領域のメモリーマップ

(c)RAM 領域のメモリーマップ

可変(RAM)領域のメモリーマップは以下のとおりである

RS303MR／RS304MD のメモリーマップ

領域	ｱﾄﾞﾚｽ No. 10 進	16 進	初期値	名称	内容	R/W
RAM 領域	30	1EH	00H	Goal Position L	指示位置	RW
	31	1FH	00H	Goal Position H	指示位置	RW
	32	20H	00H	Goal Time L	指示時間	RW
	33	21H	00H	Goal Time H	指示時間	RW
	34	22H	00H	Reserved	予備	-
	35	23H	64H	Max Torque	最大トルク	RW
	36	24H	00H	Torque Enable	トルク ON	RW
	37	25H	00H	Reserved	予備	-
	38	26H	00H	Reserved	予備	-
	39	27H	00H	Reserved	予備	-
	40	28H	00H	Reserved	予備	-
	41	29H	00H	Reserved	予備	-
	42	2AH	00H	Present Posion L	現在位置	R
	43	2BH	00H	Present Posion H	現在位置	R
	44	2CH	00H	Present Time L	現在時間	R
	45	2DH	00H	Present Time H	現在時間	R
	46	2EH	00H	Present Speed L	現在スピード	R
	47	2FH	00H	Present Speed H	現在スピード	R
	48	30H	00H	Present Current L	現在負荷	R
	49	31H	00H	Present Current H	現在負荷	R
	50	32H	00H	Present Temperature L	現在温度	R
	51	33H	00H	Present Temperature H	現在温度	R
	52	34H	00H	Present Volts L	現在電圧	R
	53	35H	00H	Present Volts H	現在電圧	R
	54	36H	00H	Reserved	予備	-
	55	37H	00H	Reserved	予備	-
	56	38H	00H	Reserved	予備	-
	57	39H	00H	Reserved	予備	-
	58	3AH	--	Reserved	予備	-
	59	3BH	--	Reserved	予備	-

( ) RS304MD の場合

Figure II-iii RAM 領域のメモリーマップ

・2 バイト長データの保存方法

メモリーマップにおいて 2 バイト長のデータを保管するときは, H(High byte), L(Low byte)それぞれ 8bit に分けて保管する.

例)ID:23 のサーボに 29.2 度動作指示を与える.

指示角度は Goal Position という項目に保存される．角度を 10 倍した整数値で操作指示を与える．そのため，10 進数で 292 だがこれを 16 進法に直すと 0124H となるので，保管されるデータは以下のようになる.

Goal Position(L) = 24H
Goal Position(H) = 01H

III. 外部ライブラリを利用した Java アプリ設定

Windows 上で動作する Java アプリケーションを外部ライブラリを利用してシリアル通信を行う方法として RXTX の導入が挙げられる．ここでは，この RXTX の導入法について記載する．

始めに RXTX ライブラリを Using RXTX のページ[12]よりダウンロードする．本研究ではWindows7 64bitを利用していたため, ダウンロードページの下部にある「x64 Binaries」にあるリンクをクリックする．

リンク先の Downloads の中にある「Windows-x64」から
ch-rxtx-2.2-20081207-win-x64.zip をダウンロードする．

ダウンロードした zip ファイルを解凍し，ライブラリをインストールする．解凍先のフォルダ内にある以下の 3 ファイル

・rxtxSerial.dll
・RXTXcomm.jar
・RXTXcomm.jar

を Install.txt を参考にしながら Java のパスの通っている場所へコピーする．

まず，RXTXcomm.jar を C:¥Program Files¥Java¥jdk1.6.0_25¥jre¥lib¥ext にコピーする．次に RXTXcomm.jar と rxtxSerial.dll を
C:¥Program Files¥Java¥jdk1.6.0_25¥jre¥bin にコピーする．以上で RXTX ライブラリのインストールは完了した．これにより，Java アプリケーションでのシリアル通信が可能となる．

Ⅳ. JNI 導入

JNI を利用する準備を以下に挙げる.

(a)Cygwin のダウンロード

Android で JIN を利用するには Android NDK(Native Development Kit)のツールの一つである NDK ビルドを行う必要があるが, これは UNIX 環境で行う必要があった. そこで UNIX 環境を Windows 上で操作できるツールとして Cygwin を利用した. 以下のリンクから Cygwin をダウンロードする[13].

ダウンロードされたファイルをインストールする. インストールする際, ダウンロードパッケージの選択で, Search 欄に「make」と入力し,「Devel」にある,

```
make: The GNU version of the 'make' utility
```

の項目で「Skip」をクリックし,「Skip」がバージョン表記になるようにする. Figure IV-i

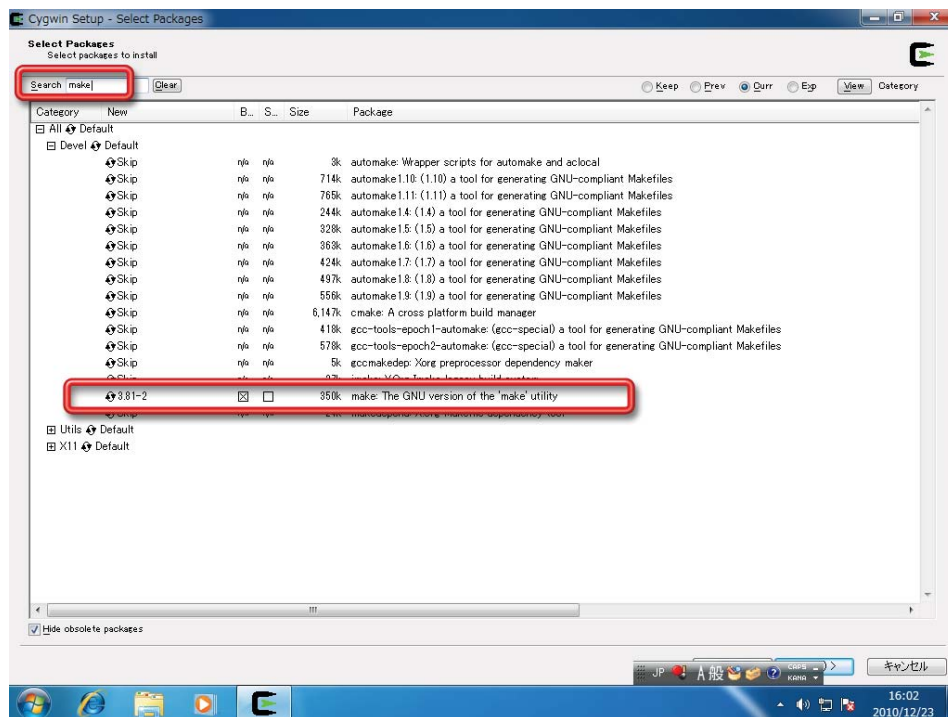



Figure IV-i Cygwin の設定(1)[14]

次に「Search」欄に「gcc4」と入力し,「Devel」にある「gcc4: GCC Release series 4 compiler (C & C++ install helper」の「Skip」をクリックして同じ様にバージョン表記になるようにする.

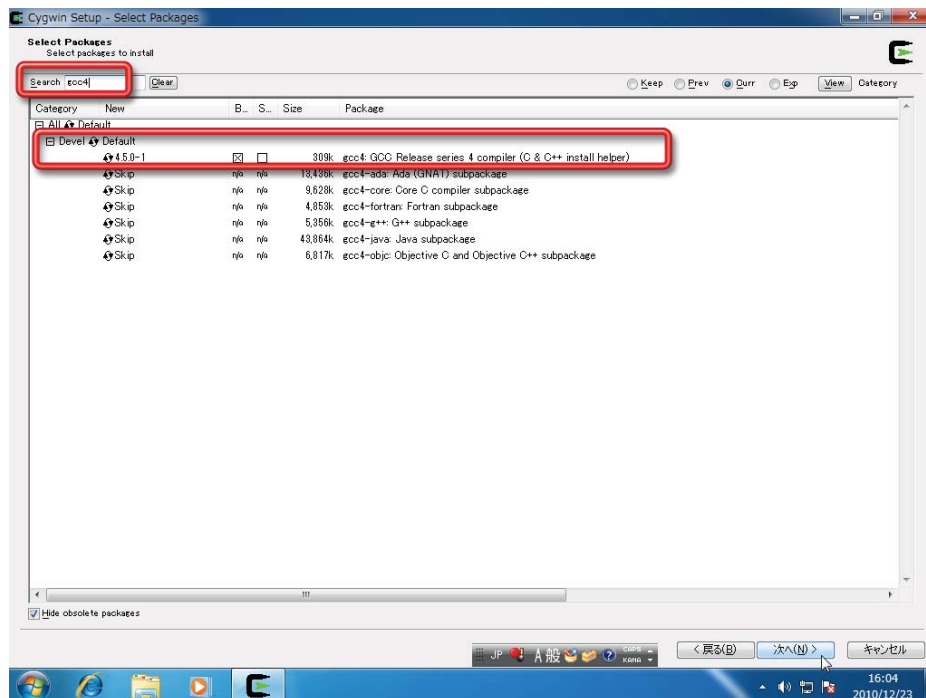



Figure IV-ii Cygwin の設定(2)[14]

(b)Android NDK のダウンロード

下記 URL より Android NDK「android-ndk-r5-windows.zip」をダウンロードする(Figure IV-iii)．r5 はバージョンであり現在のバージョンは r7 である．

http://developer.android.com/sdk/ndk/index.html

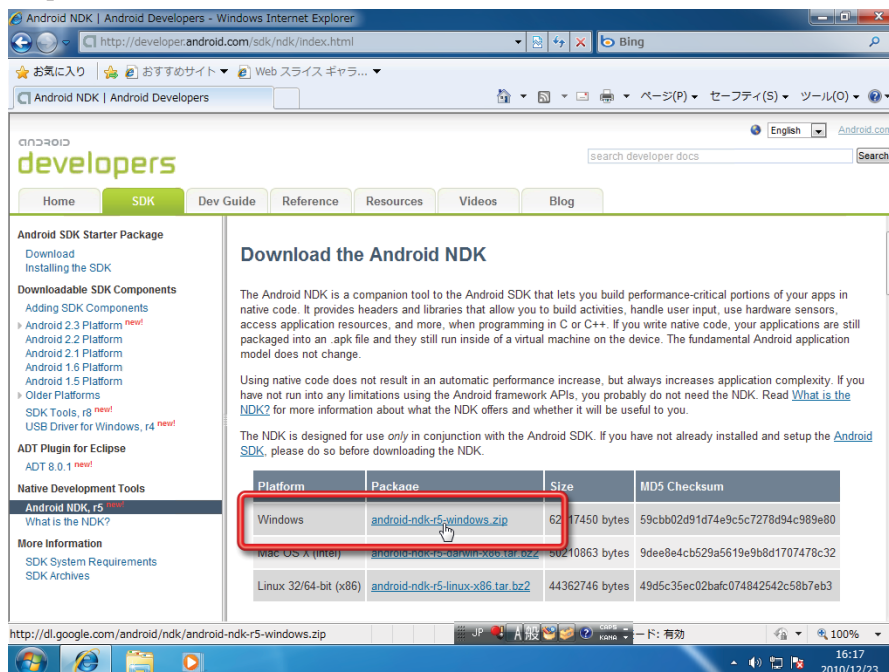



Figure IV-iii android-ndk のダウンロード[14]

ダウンロードしたファイルを展開し，C:¥直下に移動し，フォルダ名を「android-ndk」に変更する．Cygwin のインストール法については UsefullCode.net[14]を参考にし，Figure Ⅳ-ⅰから Figure Ⅳ-ⅲはこのホームページより引用している．

(c)Cygwin でユーザー用スクリプトの作成

Cygwin を起動する．起動するとユーザー用のスクリプトが自動で生成される．

(d)eclipse の設定(1)

eclipse を起動しメニューバーの「ファイル」→「開く」を選択.
Cygwin 起動で作成された C:/Cygwin/home/(ユーザー名)/.bashrc を開き，その末尾に以下の 2 行を追加する．

```
export ANDROID_NDK_ROOT=/cygdrive/c/android-ndk
export PATH=$PATH:$ANDROID_NDK_ROOT
```

メニューバー「ヘルプ」→「ソフトウェアのインストール」を選択する．
作業対象のリストから「Helios - http://download.eclipse.org/release/helios」を選択すし，「Programming Language」の中から「C/C++ Development Tools」と「C/C++ Library API Documentation Hover Help」にチェックを入れ「次へ」を選択する．
インストール後，eclipse を再起動する．

メニューバー「ウィンドウ」→「設定」→「実行/デバッグ」→「起動」→「デフォルトランチャー」→「C/C++ Application」→「デバッグ」→「Standard Create Process Launcher」にチェックを入れる．

eclipse の設定は以上である．

(e)JNI の導入

以下は作成するアプリケーションに JNI を導入する工程である．
(1)プロジェクトの中にフォルダ「jni」を作成

(2)作成した「jni」フォルダに「ファイル」→「新規」→「ファイル」で, ファイル名「Android.mk」を作成．この際，大文字，小文字の区別に注意する．

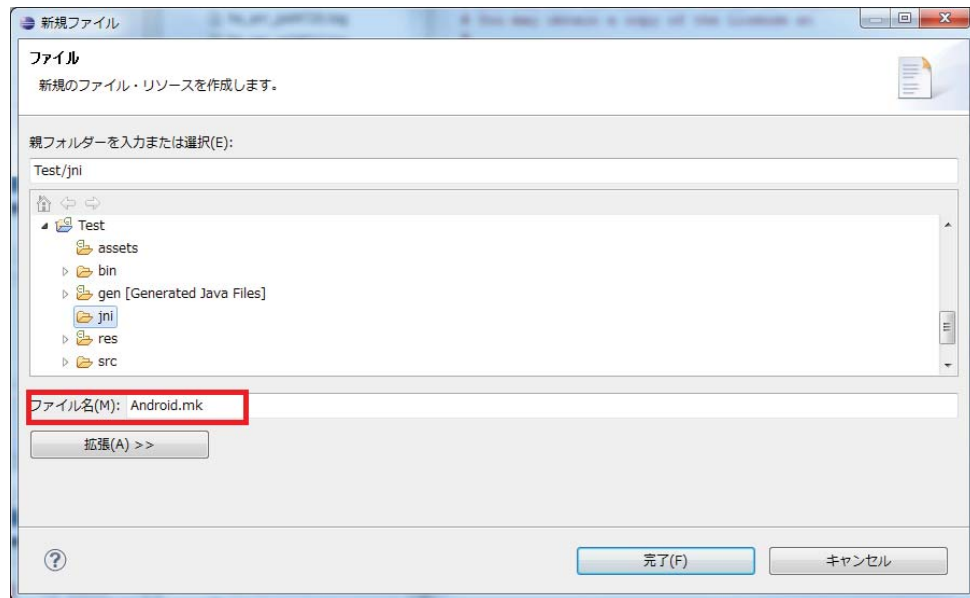



Figure IV-iv JNI の導入(1)

(3)作成した「Android.mk」を開き，以下の 5 行を書き込む．

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := SerialJNI
LOCAL_SRC_FILES := SerialJNI.c
include $(BUILD_SHARED_LIBRARY)
```

Figure IV-v は実際に Android.mk に書き込んだものである．

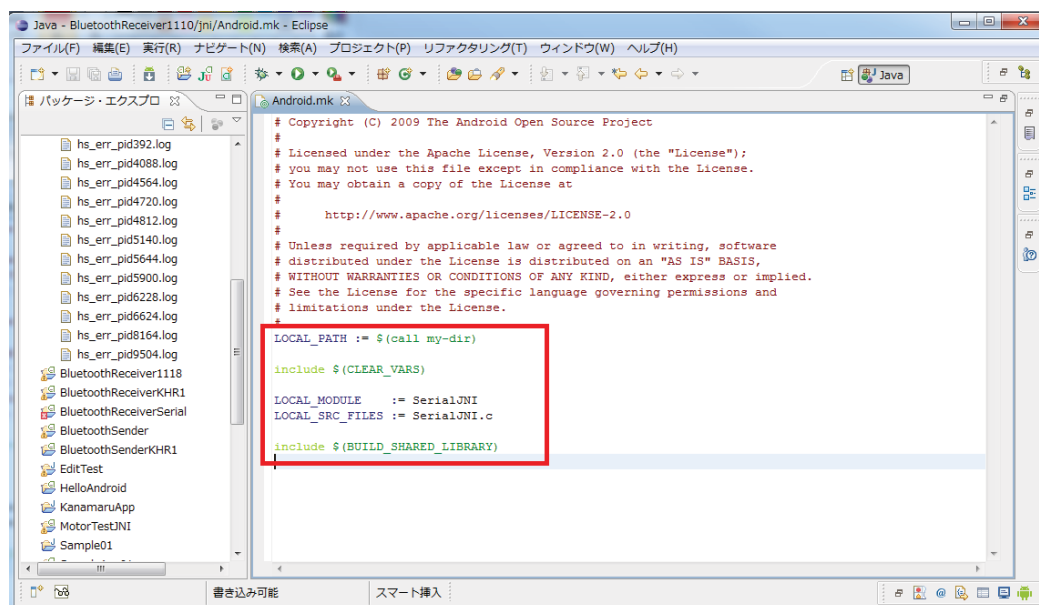
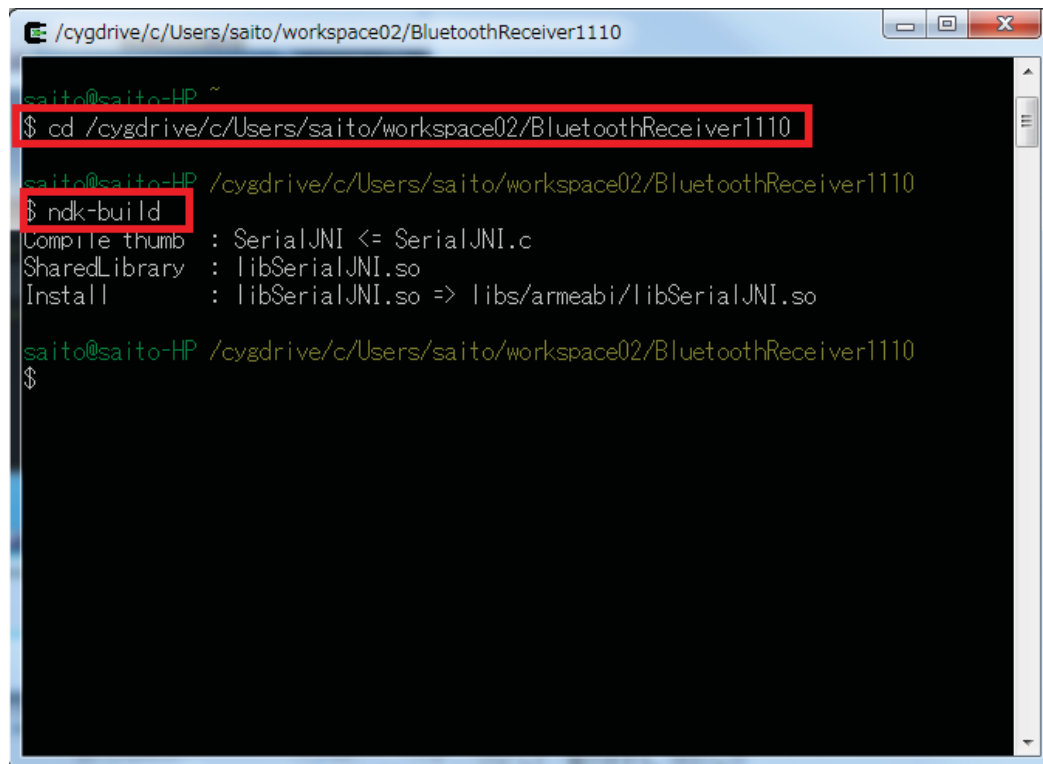



Figure IV-v JNI の導入(2)

(4)「Android.mk」を作成したフォルダに同様にファイル名「SerialJNI.c」を作成する．ここにシリアル通信のためのモジュールを書き込む．
Java へエクスポートする関数は「Java_プロジェクト名_アクティビティ名_関数名」とする．今回は
「Java_com_tk_android_BluetoothReceiverKHR1_BluetoothReceiver_openSerial」
となっている．

(5)Java ソースファイルに以下の 3 行を追加しすべてのファイルを保存する．

```
public native void openSerial(String port, int baudrate);
public native void closeSerial();
public native void writeData(byte[] buf, int n);
```

(6)Cygwin を起動し，ndk-build を行う．Figure IV-vi

Figure IV-vi Cygwin での操作

```
$ cd /cygdrive/プロジェクトの場所/プロジェクト名
(本研究では$ cd /cygdrive/c/Users/saito/workspace02/BluetoothReceiver1110)
$ ndk-build
```

カッコを除く 2 行を実行する．
以上の工程を行った後，Android のアプリを実行することで利用可能となる．

付録図表目次

Android OS を搭載した組み込みデバイスと
スマートフォンによるモータの制御

指導教員　金丸 隆志　准教授
AM-10039　齋藤 翼

1.　緒言

現在，Android OS が注目を集めている．2011 年末にはスマートフォン用 OS としてトップシェアを占めている．その理由として，Android OS はオープンソースであり，誰でも無償で利用できることが挙げられる．現在ではスマートフォンだけでなく，家電や，自動車のカーナビゲーションシステムにも利用されるようになっている．このように汎用性の高い Android の OS をスマートフォンとは異なる組み込みデバイス用 OS として利用することで新しい利用の場を提案する．本研究では，スマートフォンと Android を搭載した組み込みシステムを利用したロボットアームの遠隔操作を目指す Android では，近接センサ，温度センサなど多くのセンサが利用できる中，3 方向加速度センサと，傾きセンサを主に利用する．これらのセンサの値をスマートフォンで取得し，Bluetooth で送信し，組み込みデバイスは，受信した値でロボットアームを制御する．この際スマートフォンを持った腕と同じようにロボットアームを動作させることを目的とする．組み込みデバイスのターゲットとして TEXAS INSTRUMENTS 製の PandaBoard を利用する．

2. センサ値の取得

ロボットアームを腕と同じ操作をさせるためにはセンサの値で腕の動きが判別できなければならない．今回利用するセンサは 3 方向加速度センサと傾きセンサである．まず，これらのセンサの値から腕の動作判別ができるかを確認した．本研究では 2 つの動作パターンに焦点を当て，作業を進める．1 つ目は，腕を下に降ろした状態から前方へ約 90 度振り上げ，元の位置まで戻す動作(前振り)，2 つ目は，腕を下に降ろした状態から横に 90 度振り上げ，元の位置まで戻す動作(横振り)である(図 1).

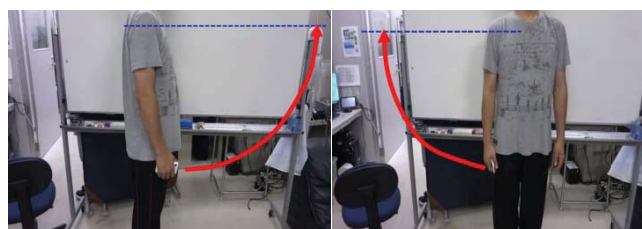



図 1 前振り(左)，横振り(右)動作

図 2 はスマートフォンで取得したセンサの値を SD カードに保存するアプリケーションである．画面中央にある ON/OFF(トグルボタン)が ON の間，センサのデータを CSV ファイルとして SD カードに保存し続ける(図 2).

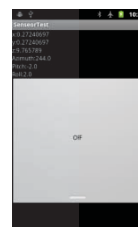



図 2 センサ値取得アプリケーション

3. センサ値の比較

取得したセンサの値を比較し，前振りのみ，横振りのみに対応した値があるかを確認する．図 3 は前振りと横振り時における加速度センサの値である．

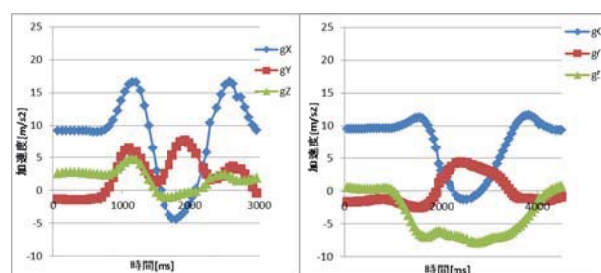



図 3 前振り(左)，横振り(右)の加速度センサ値

センサの値はすべて反応してしまっているため，加速度センサの利用だけでは腕の動作を判別することが難しい．

次に，前振り時と横振り時の傾きセンサの値の比較をする．(図 4)

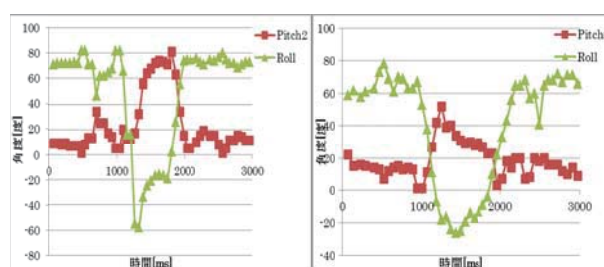



図 4 前振り(左)，横振り(右)の傾きセンサ値

こちらもセンサの値はすべて反応してしまっている．以上のことから加速度センサ，傾きセンサの値をそのまま利用することはできないことがわかる．前振りのみ，横振りのみに反応する値を取得するため，ニューラルネットワークを利用した．

4．ニューラルネットワークによる値の加工

これまで取得してきたセンサの値を更に加工し，モータの制御に利用するため，ニューラルネットワークを利用した．今回は取得しているセンサの値(加速度 3 成分，重力加速度 3 成分，動加速度 3 成分，傾き 3 成分)12 個の中から，加速度 3 成分，重力加速度 3 成分，傾き 2 成分(Pitch と Roll)の自由度を入力とし，前振りにのみ反応する値，横振りにのみに反応する値の 2 つの値，$\widetilde{oy}$と$\widetilde{oz}$を出力として得る．出力を獲るために利用した教師データは手作業で作成した．図 5 はニューラルネットワークで値を加工する工程である．ここで，傾き成分ox，oy，ozは，それぞれ Azimuth，Pitch，Roll に対応し，$\widetilde{oy}$は Pitch2 に対応している．

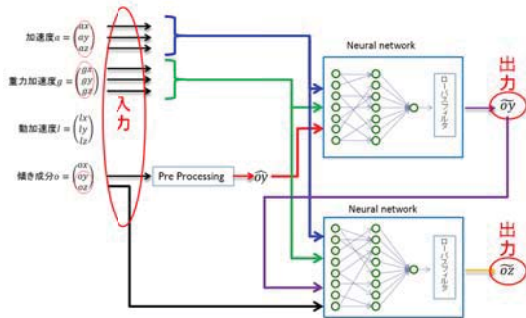



図 5 ニューラルネットワークによるセンサ値の加工

ニューラルネットワークにより加工された前振り，横振りの結果は図 6 のようになった．

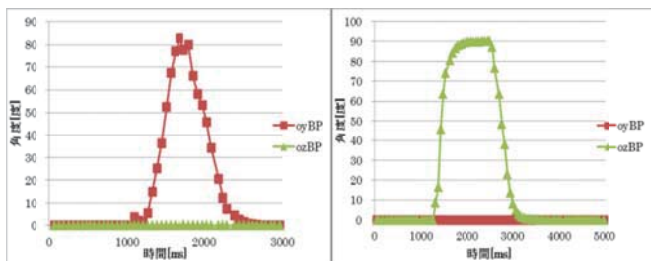



図 6 加工後の前振り(左)，横振り(右)データ

これにより，前振りと横振りのそれぞれのみに反応する値を取得することができた．この結果を用いて Android 搭載のスマートフォン，PandaBoard を利用したモータ制御を行う．

5.BluetoothSender

BluetoothSender はニューラルネットワークで得た 2 つの値とサーボモータの ID を Bluetooth を用いて PandaBoard に送信するアプリケーションである．本アプリケーションは，左右両方の手に対応させ，メニューボタンにて変更ができる．また，本研究の目的であるロボットアームを腕と同じ動きをさせる「大振り版」とは別に，スマートフォンを水平に持った手を前後左右に傾けるだけで操作する「小振り版」のアプリケーションも組み込んだ．「大振り版」，「小振り版」の切り替えは面上にあるタブにて切り替えができる．

5．BluetoothReceiver

BluetoothReceiver はスマートフォン用アプリケーションである BluetoothSender から送られた値を基にロボットアームを制御するためのアプリケーションである．ロボットアームの制御にはシリアル通信を利用する．シリアル通信をする際，アプリケーションからハードウェアにアクセスする必要がある．しかし，Android アプリケーションは Java ベースで作成するため直接ハードウェアにアクセスすることはできない．そこで，JNI(Java Native Interface)を利用する．JNI は Java のプログラムと C，C++で書かれたコードを連携するためのインターフェースである．JNI の特徴として「ハードウェアにアクセスすることができる」こと，「システムを速くすることができる」ことなどが挙げられる．今回は前者を目的として利用することで Java からハードウェアへのアクセスが可能になる． BluetoothSender，BluetoothReceiver を用いて実際にアームの制御を行ったところ，腕の動作にやや遅れるが，ロボットアームも動かした腕と同じ動作をすることが確認できた．

6．結言

本研究では，スマートフォンのセンサ値を用いたロボットアームの遠隔操作アプリケーションを作成した．スマートフォンの 3 方向加速度センサ，傾きセンサの値を利用することにしたが，センサ値は前振り，横振り動作時に全てのセンサの値が反応してしまった．前振りのみ，横振りのみに反応する値を作成するため，バックプロパゲーションによる値の加工を行い，それぞれの動作のみに反応する値を得ることができ，スマートフォンをも持った腕と同じ動作をさせることが可能になった．しかし，アームの動作は，腕の動きに少し遅れて動作する．これはニューラルネットワークを通した際に発生した時間のずれである(図 7)．

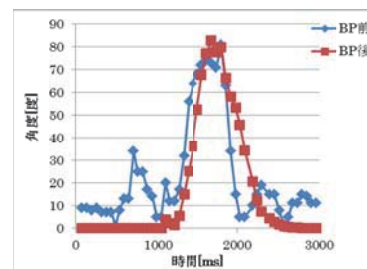



図 7 ニューラルネットワーク前後の比較

この間隔に対してはニューラルネットワークに加える入力のより良い組み合わせを見つけることで，アーム動作の遅れを更に減らすことができるのではないかと考えている．